PETUUM

Carnegie Mellon University
School of Computer Science

Composable Machine Learning

Eric Xing
Petuum & Carnegie Mellon

ODSC West 2019

Acknowledgements

Zhiting Hu
Scientist @Petuum Inc.
PhD Candidate @CMU

Qirong Ho
Co-founder, CTO @Petuum Inc.

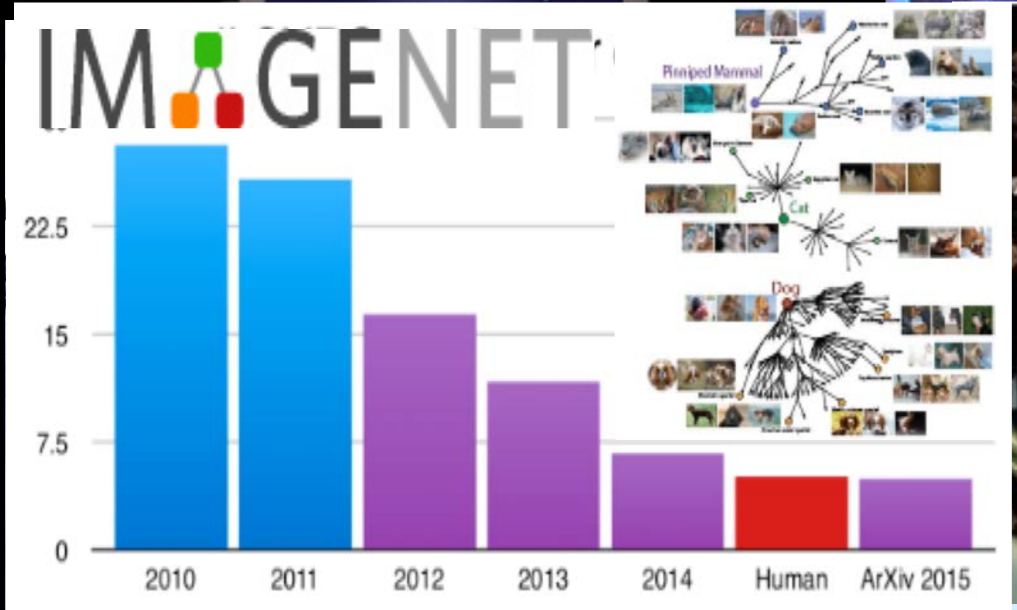
R&D team @ Petuum

Sailing Lab @ Carnegie Mellon University

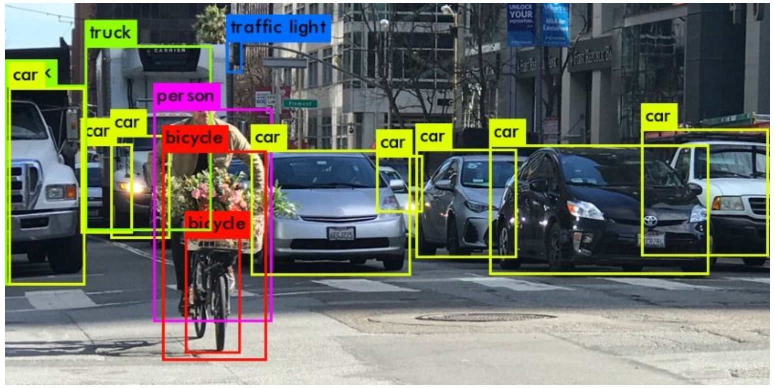
An AI era?

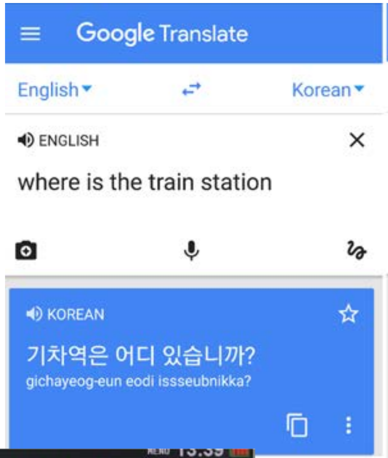

AlphaGo
Lee Sedol

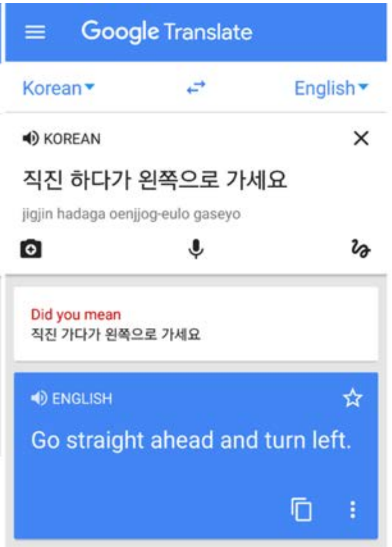

THINK
$300,000
$1,000,000
$200,000
$17,973


IMAGENET
22.5
15
7.5
0
2010
2011
2012
2013
2014
Human
ArXiv 2015
Pinniped Mammal
Cat
Dog

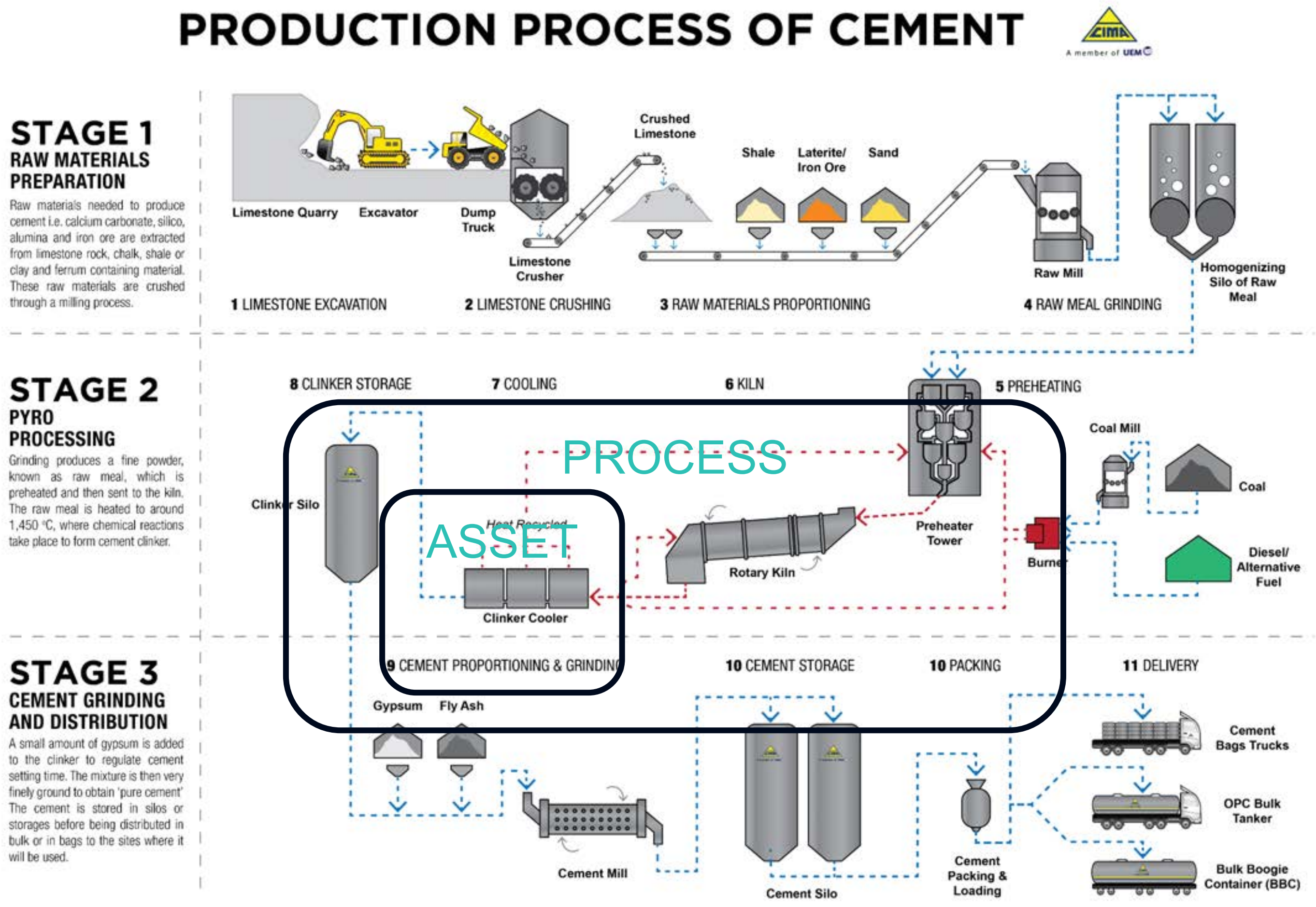
Real-world Machine Learning Problems







The Cement Industry

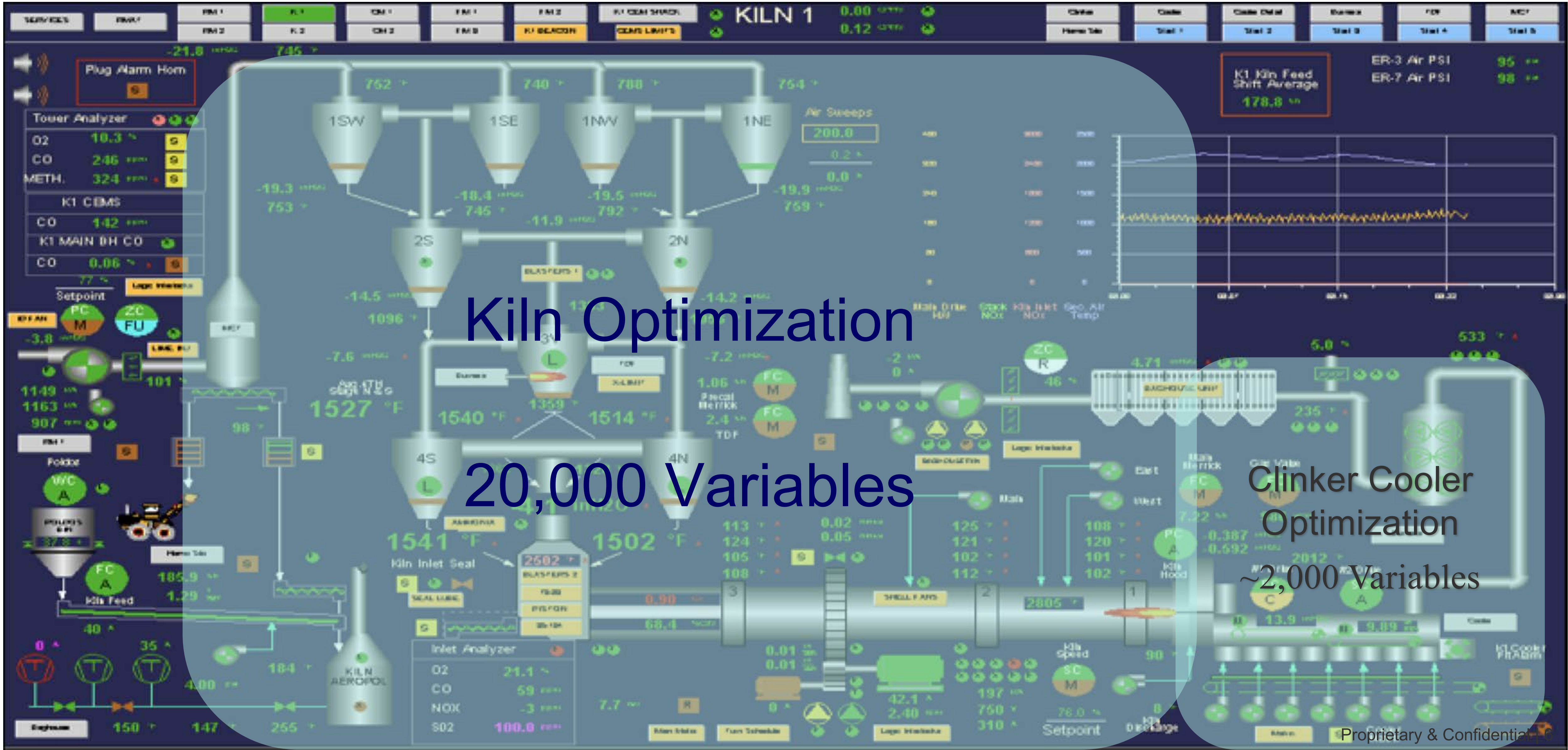


Operational Excellence Initiatives

- Emissions (NOx, SOx)
- Fuel-Mix (Traditional vs. Renewables vs. Alternatives
- Benchmarking fleet Performance
- Predictive/Preventive Maintenance

Petuum

Assess, Process, Factory, Corporate Optimization

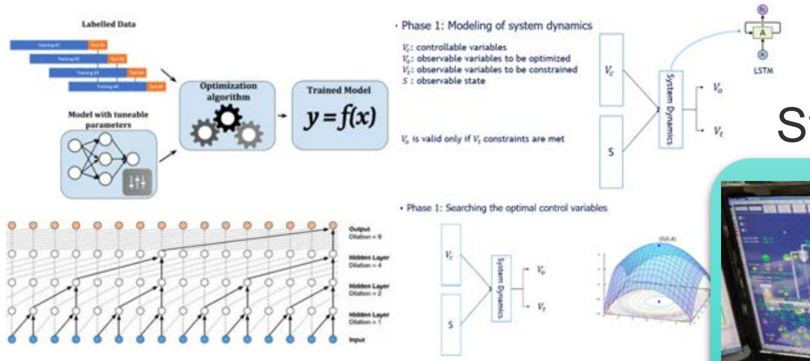


Building an ready-to-use AI solution for this is

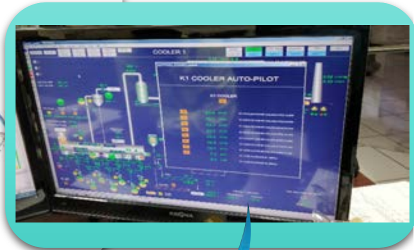
Extremely complex

Raw data far from ideal

Simultaneous Prediction and Control



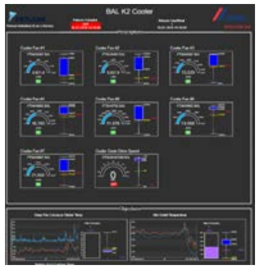
Systems/Infra



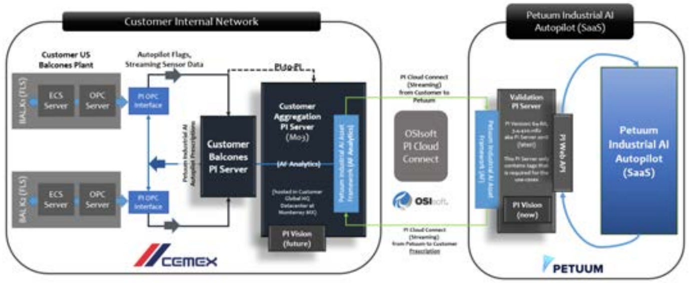
Requires inter-operation between diverse systems

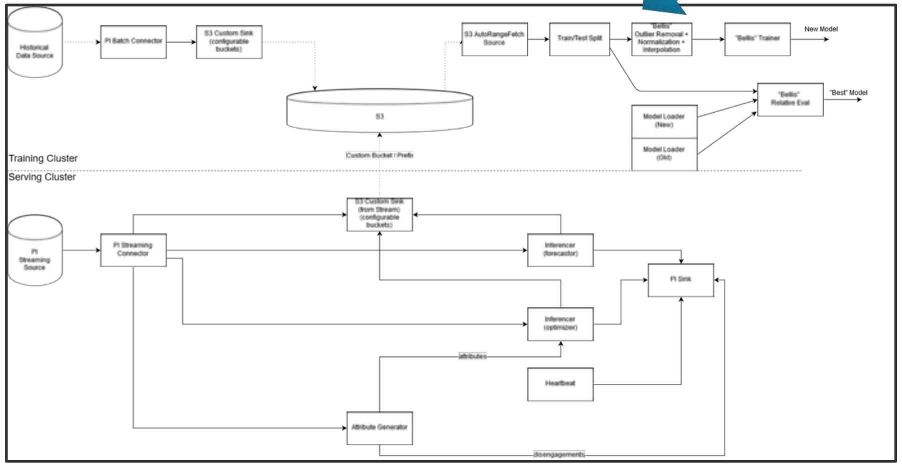
Can't solve with "algo marketplace" or Kaggle competition

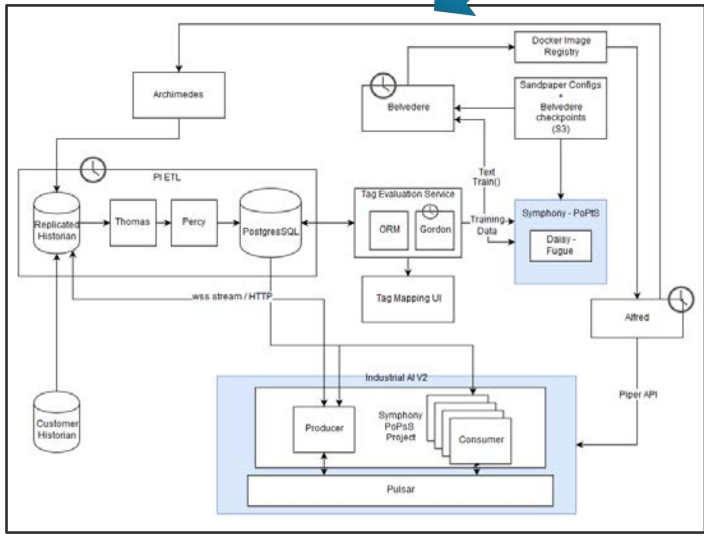
Dashboard



Factory IT Infrastructure





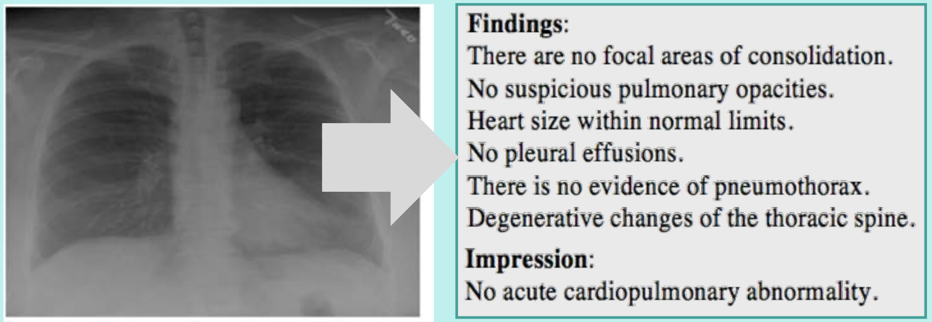


Petuum

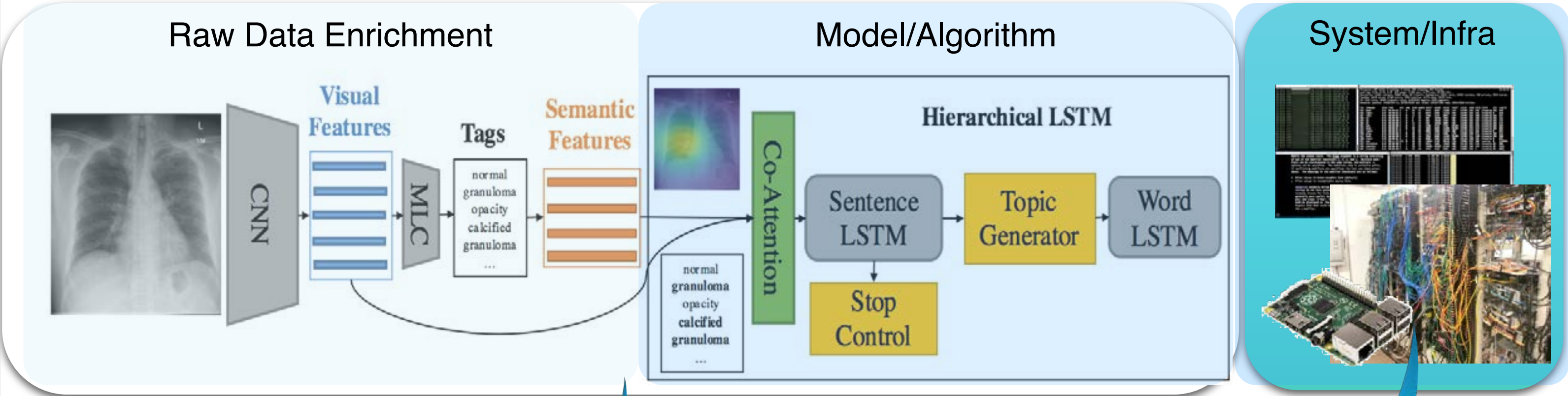
The Healthcare Industry

Building an ready-to-use AI solution for this is

Extremely complex



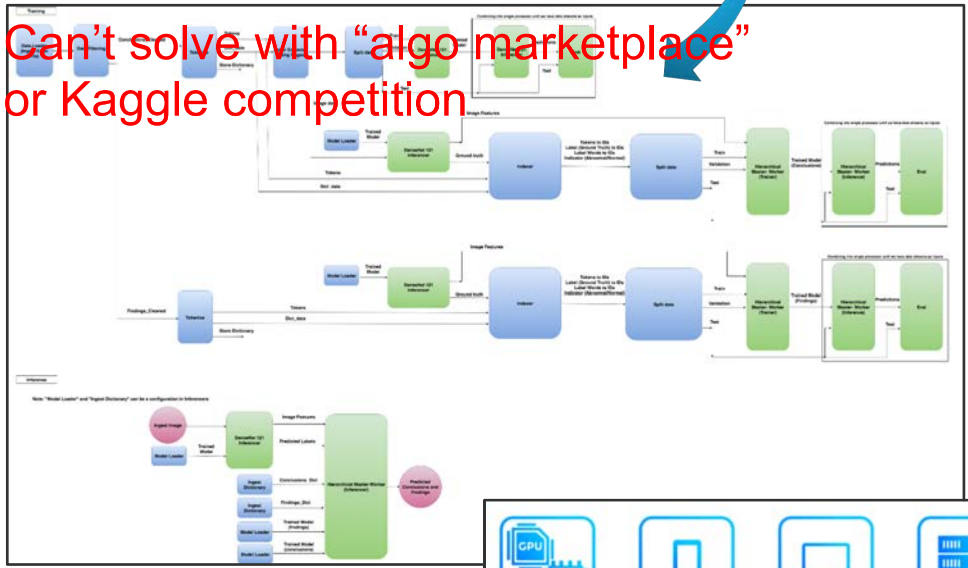
Task: Automatic Medical Report Generation

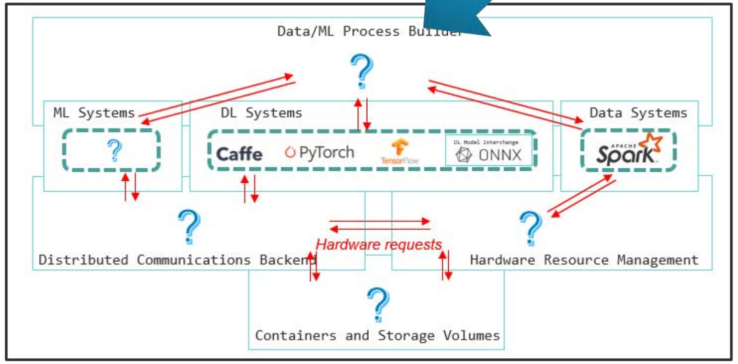


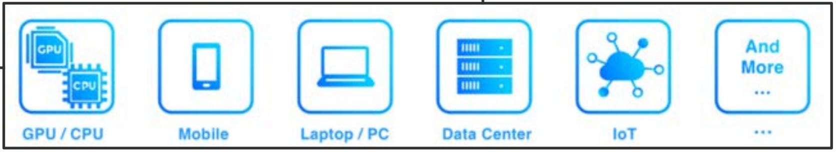
Requires inter-operation between diverse systems

User interface for Doctors

Can't solve with "algo marketplace" or Kaggle competition







AI as of now: Not Built, but Crafted

Petuum

Outline

- Composable ML

  A first-principle view of ML components and assemblage

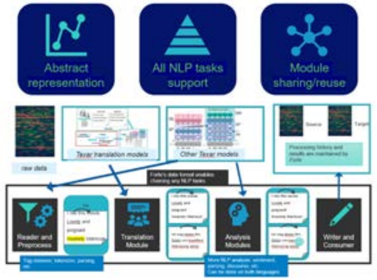
- Texar: A Modularized ML toolkit

  Compose your ML applications like playing building blocks

Texar

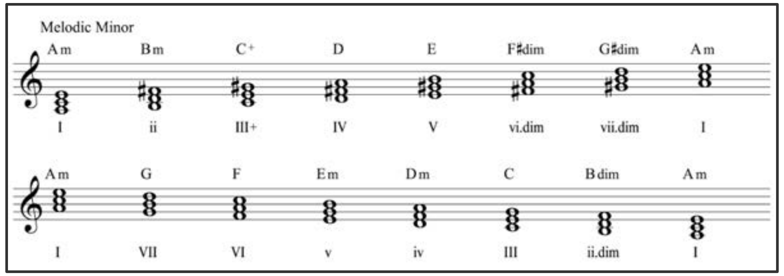
- Scalable AI Infrastructure

  Composable ML in production



Composable ML

One-off design and programming

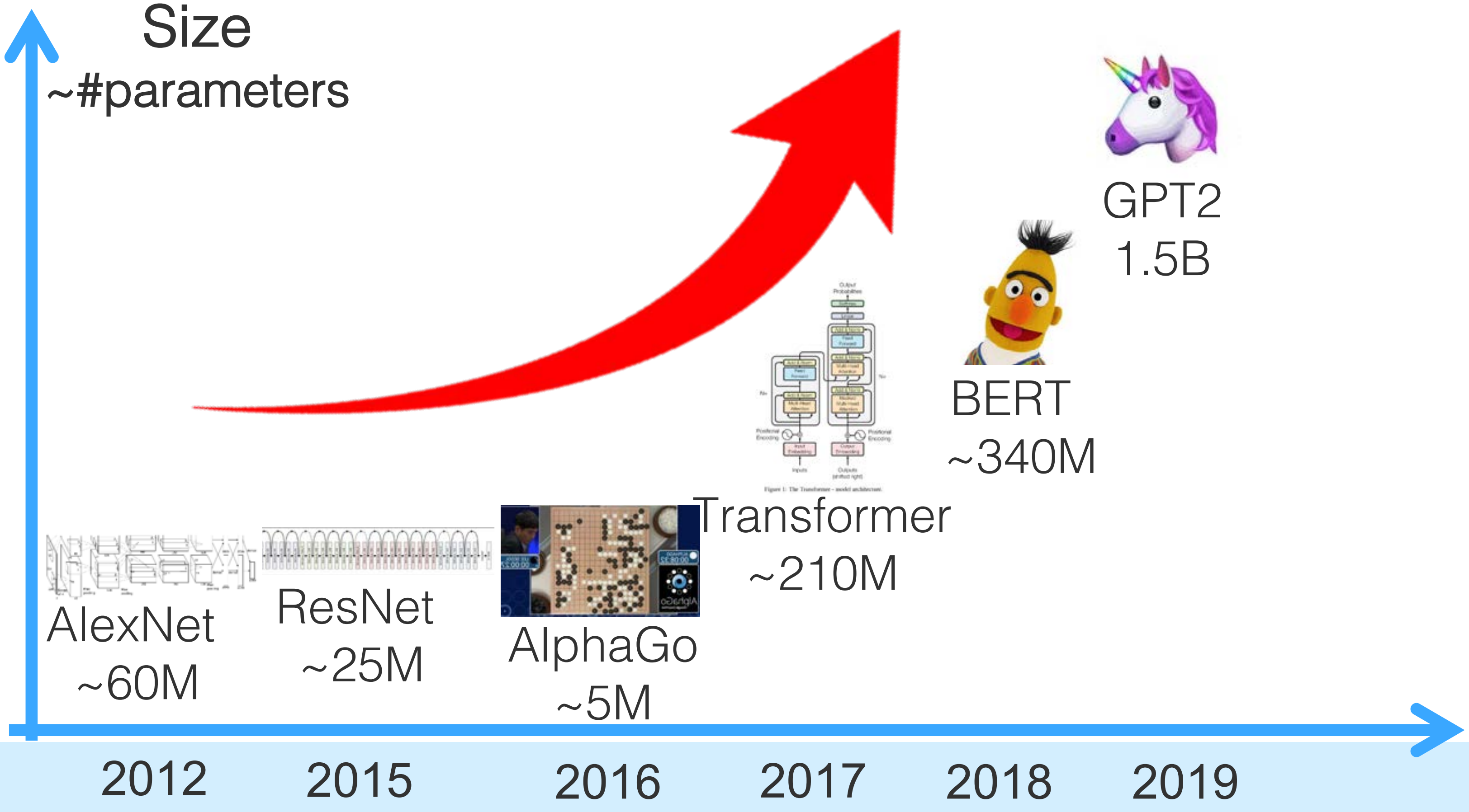


Modular and standardized

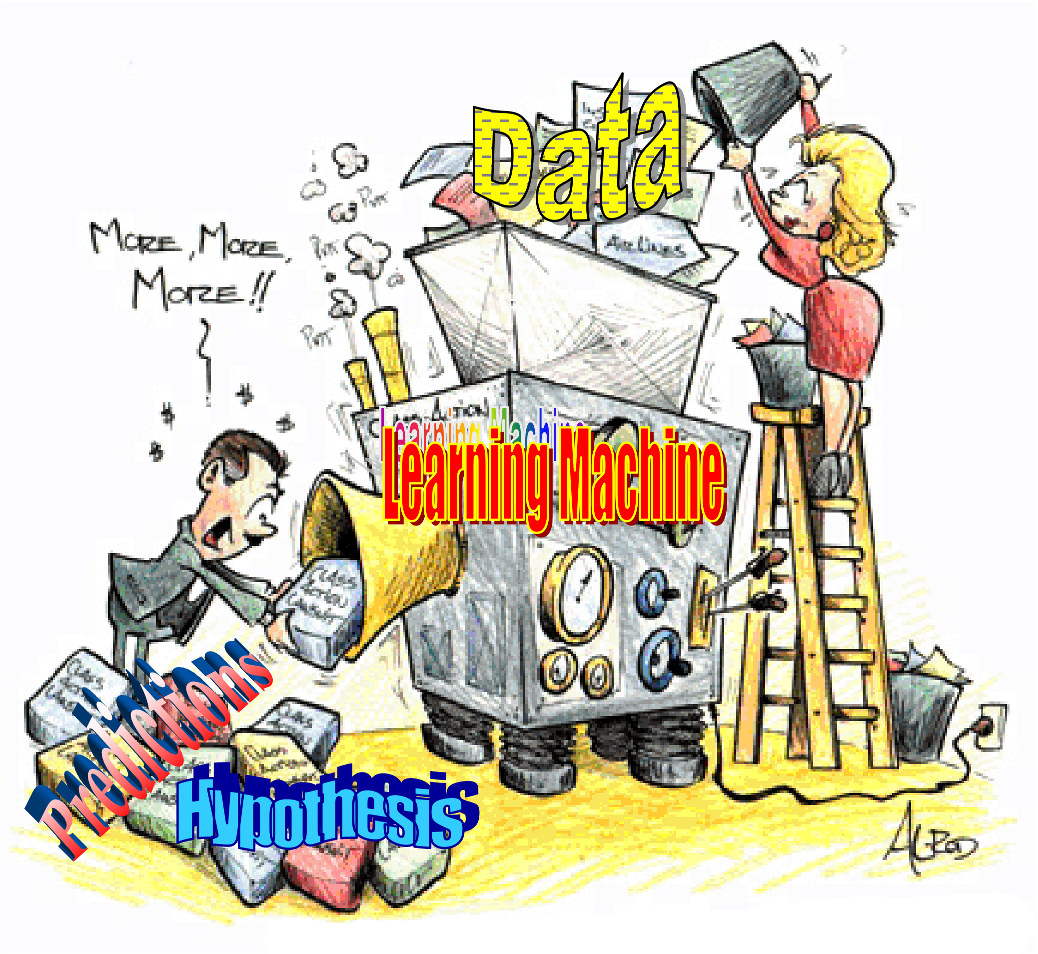
Complex “rhythms” and “chords”
systematically built out of simpler “notes”

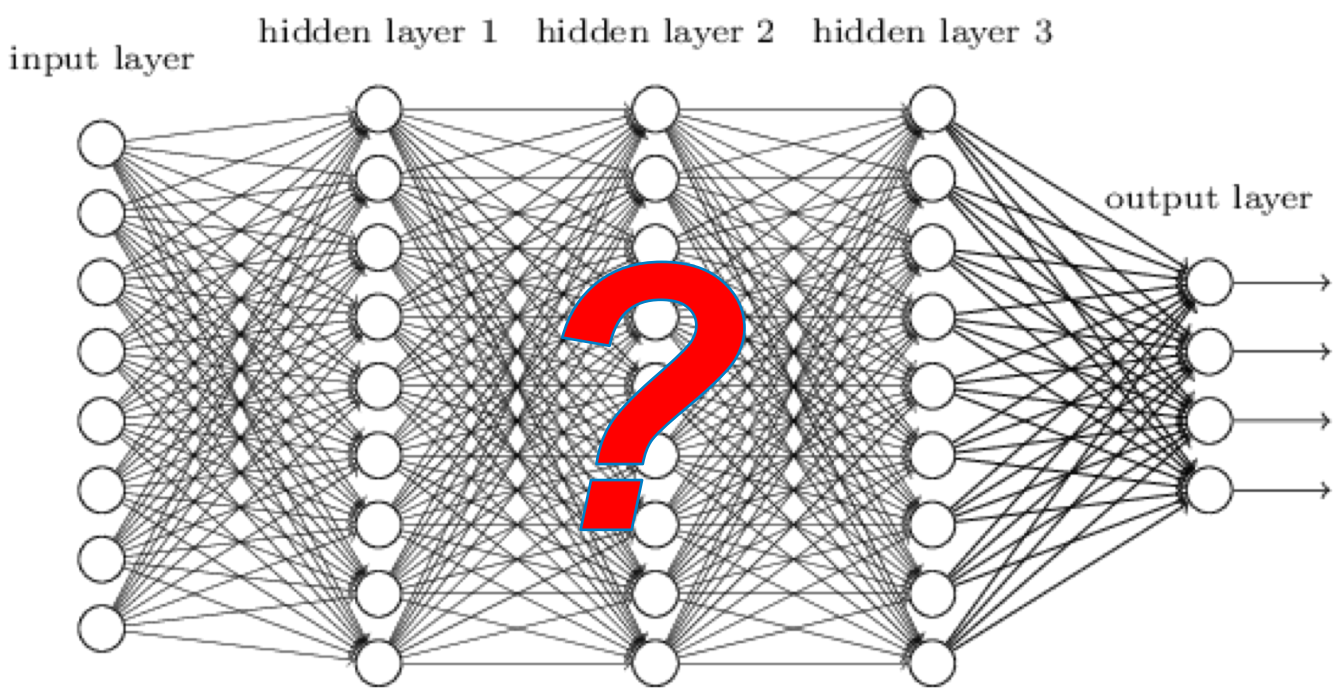
Single Models with Increasing Size and Performance

- Increasingly large black-box neural networks
- Good, even super-human performance on some tasks

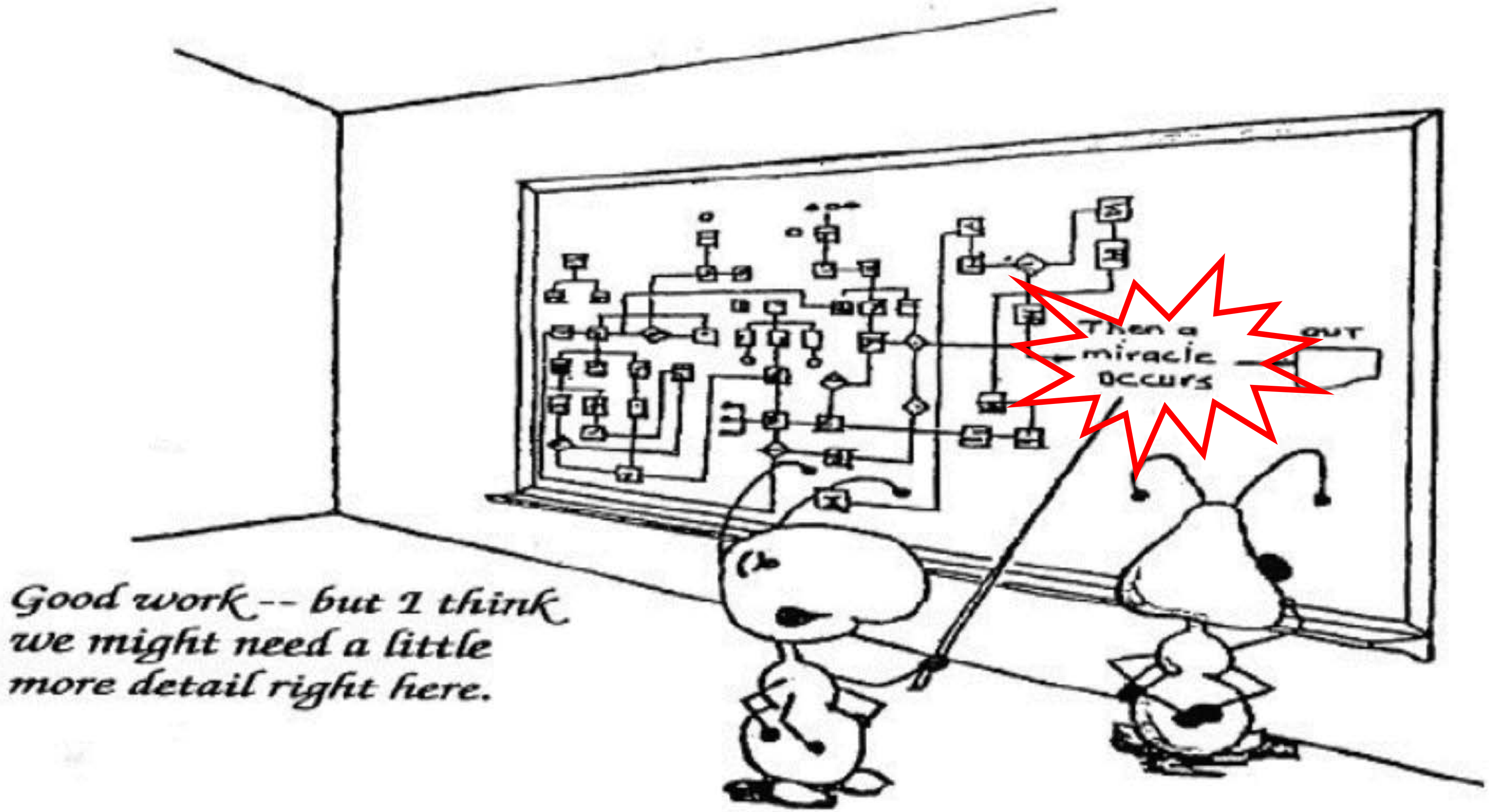


Single Giant Models Enough?



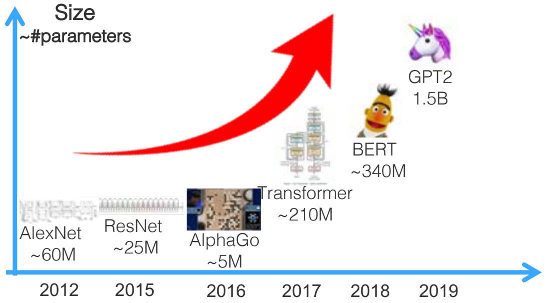


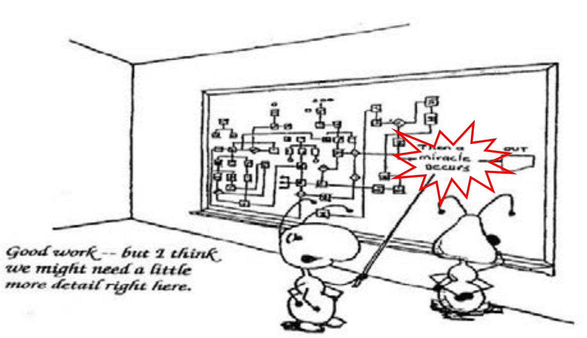
Difficulties of Single Giant Models



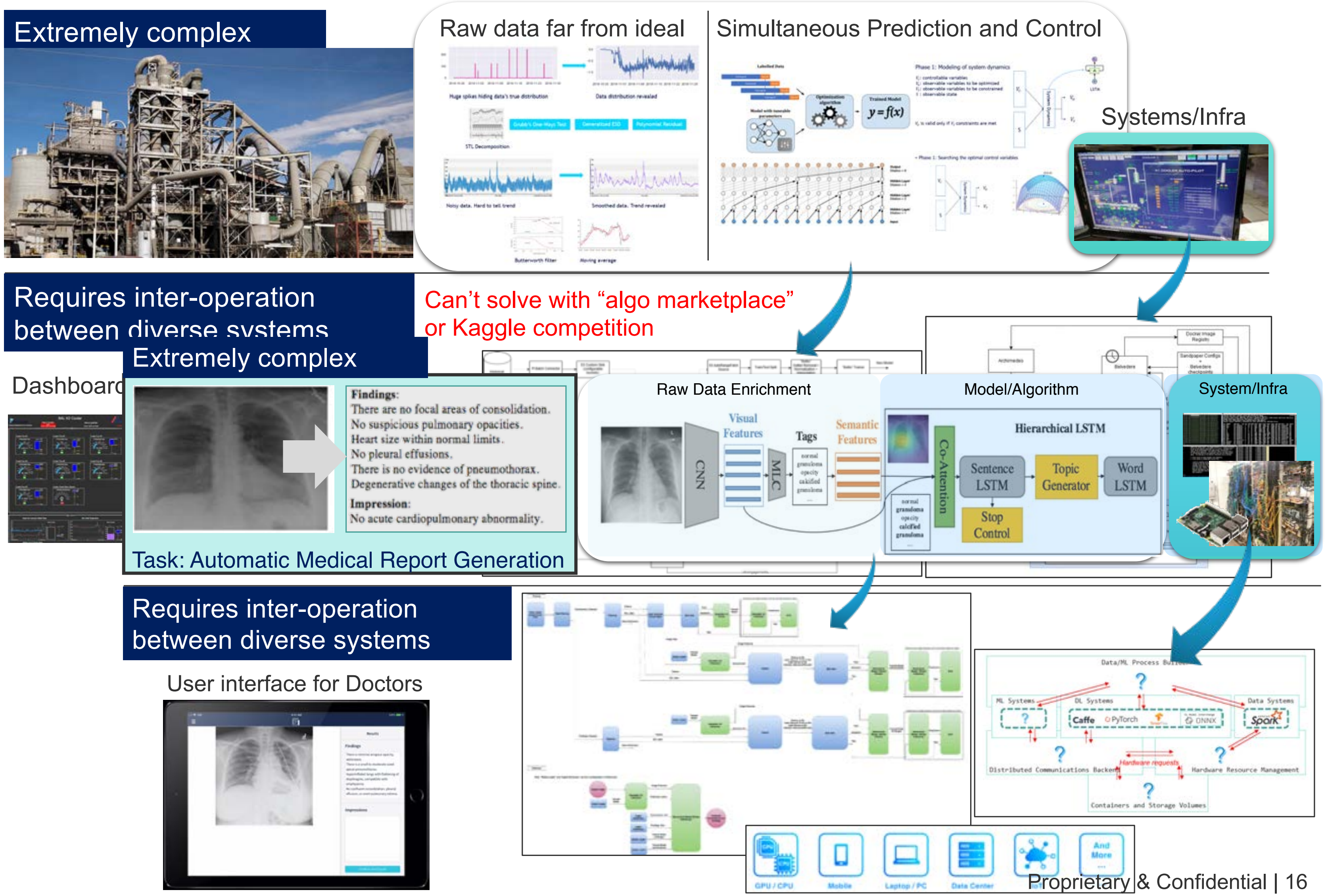
- Explainability
- Expertise
- Debugging
- Maintenance
- Upgrade
- Scalability
- …

Far from Solving Real Complex Problems





V.S.



Petuum

We choose MT as our running example to keep this talk to 90min

Running Example: Machine Translation (MT)

raw data

cleaning
tokenizing
vocabulary
truncation
…

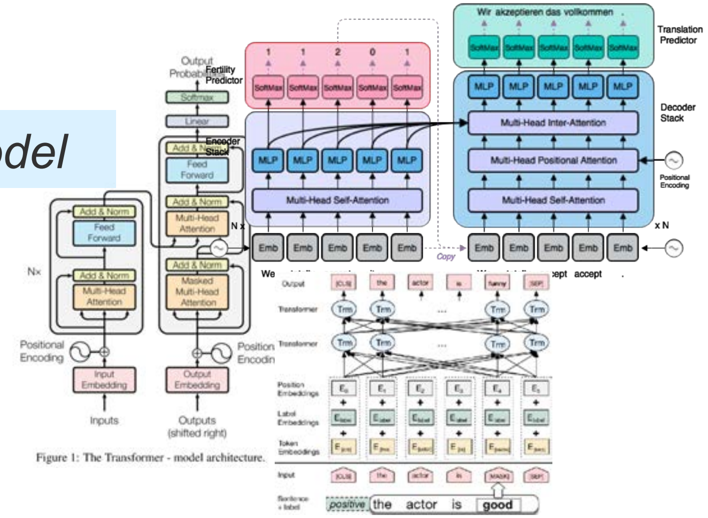
source.dat

I like this movie.
Lovely and poignant
Insanely hilarious!
…

target.dat

Ich mag diesen film.
Schön und ergreifend
Wahnsinnig witzig!
…

clean data



model

training

Maximum likelihood training

Reinforcement learning

Adversarial

Finetuning

evaluation
post-processing

…

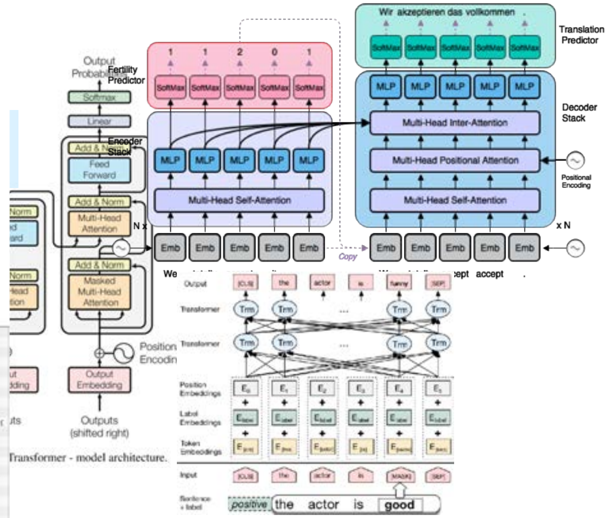
Petuum

Running Example: Machine Translation (MT)

raw data

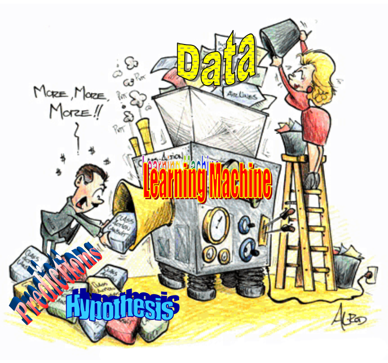
evaluation post-processing

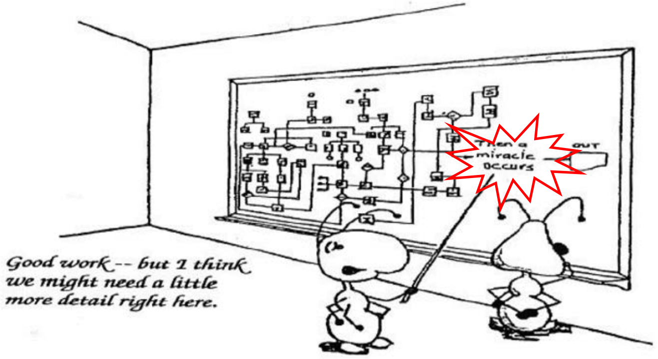
...



Petuum

Solution: Composable ML





Composable ML

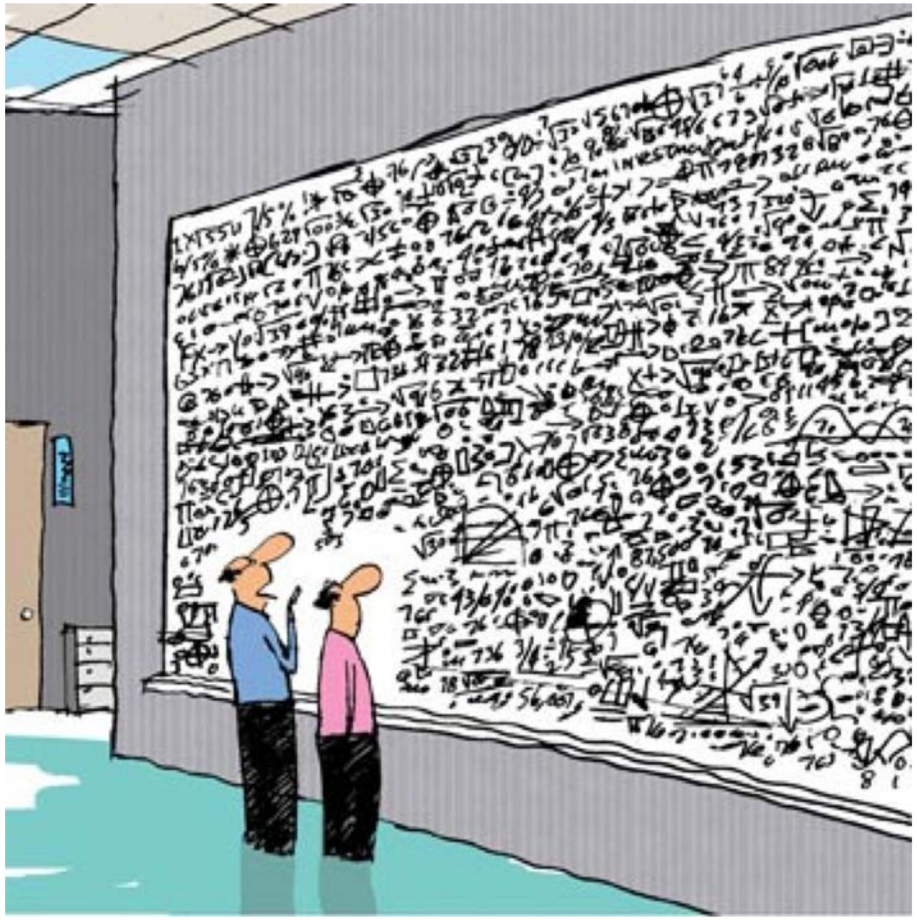
A **modularized** way to build complex applications

Petuum

Solution: Composable ML

- Build AI solutions more easily, via pick-and-choose:
  - Data: text, speech, image, video, time series, …
  - Models: recurrent, transformer, convolutional, …
  - Tasks: classification, regression, generation, discovery, …
- Stop writing same one-off code again and again
  - More reliable and easier to debug
  - Easier to onboard new developers

First Principles: Decomposing Machine Learning



Loss functions
(likelihood, reconstruction, margin, …)

Model architectures
(RNNs, Transformers, Graphical, …)

Constraints
(normality, sparsity, logical, KL, sum, …)

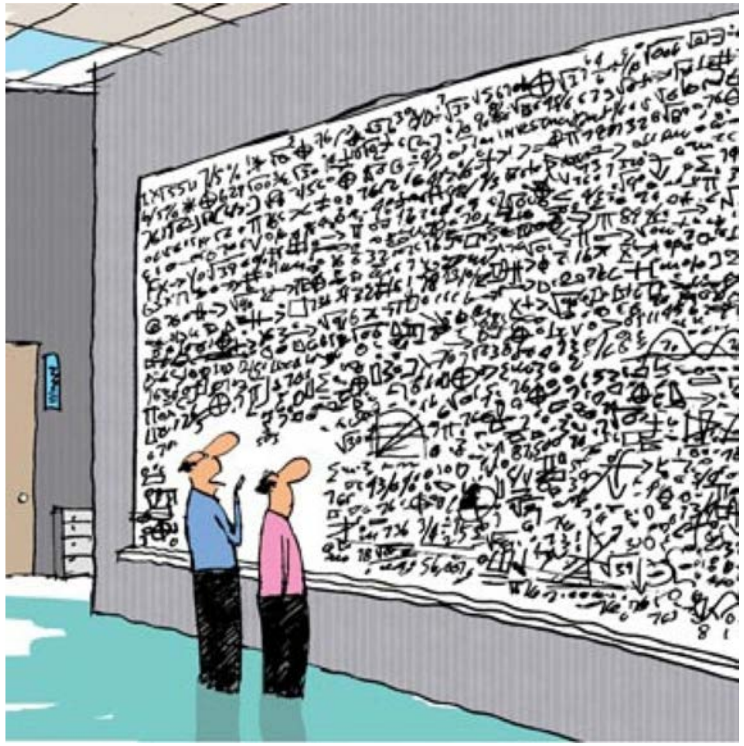
Algorithms
MC (MCMC, Importance), Opt (gradient, IP), …

Data
(processing, augmentation, weighting, ...)

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience



$$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$$

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience

$$\boldsymbol{y} \sim p_\theta(\boldsymbol{y}|\boldsymbol{x})$$

$$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$$

model architecture/
inference procedure

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$$

$(\boldsymbol{x}^*, \boldsymbol{y}^*)$

experience (data)

model architecture/
inference procedure

Petuum

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$$

loss

model architecture/ inference procedure

experience (data)

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$$

constraint

experience (data)

loss

model architecture/
inference procedure

Decomposing Machine Learning

Machine Learning:

Computational methods that enable machines to learn concepts and improve performance from experience

$$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$$

constraint

learning procedure

experience (data)

loss

model architecture/
inference procedure

$$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$$

Running Example: Machine Translation

raw data

cleaning
tokenizing
vocabulary
truncation
…

source.dat

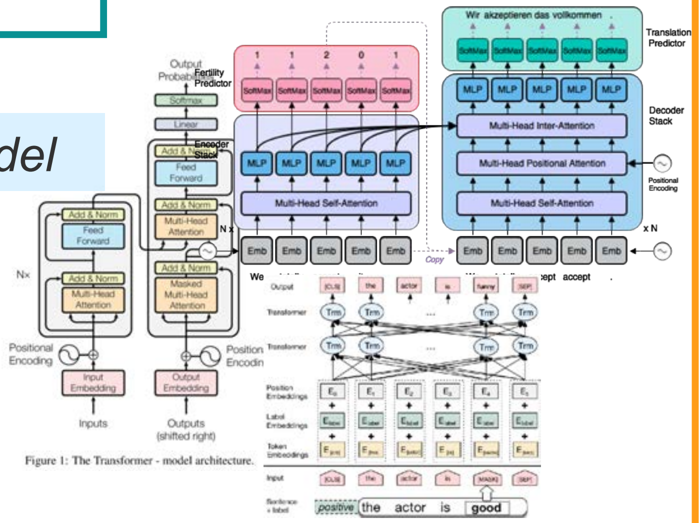
I like this movie.
Lovely and poignant
Insanely hilarious!
…

target.dat

Ich mag diesen film.
Schön und ergreifend
Wahnsinnig witzig!
…

clean data

model



training

Maximum likelihood training

Reinforcement learning

Adversarial

Finetuning

evaluation
post-processing

…

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint

Loss

Learning

Inference

Architecture

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint

Loss

Learning

Inference

Architecture

Architecture (1): Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Calculates the probability of a sentence:
  - Sentence:

$$\boldsymbol{y} = (y_1, y_2, \ldots, y_T)$$

Example:

(I, like, this, …)

Architecture (1): Language Model

$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$

- Calculates the probability of a sentence:
  - Sentence:

$$\boldsymbol{y} = (y_1, y_2, \ldots, y_T)$$

$$p_\theta(\boldsymbol{y}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1})$$

Example:

(I, like, this, …)

$$\cdots p_\theta\,(like \mid I)\, p_\theta(this \mid I, like) \cdots$$

Architecture (1): Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Calculates the probability of a sentence:
  - Sentence:

$$\boldsymbol{y} = (y_1, y_2, \ldots, y_T)$$

$$p_\theta(\boldsymbol{y}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1})$$

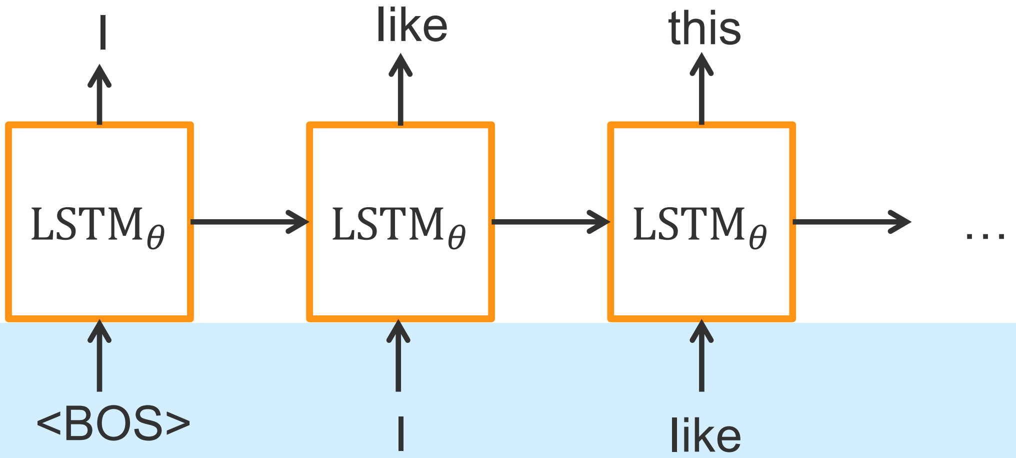
Example:

(I, like, this, …)

$$\cdots p_\theta\,(like \mid I)\; p_\theta(this \mid I, like) \cdots$$

Architecture (1.1)

LSTM RNN



Architecture (1): Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Calculates the probability of a sentence:
    - Sentence:

$$\boldsymbol{y} = (y_1, y_2, \ldots, y_T)$$

$$p_\theta(\boldsymbol{y}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1})$$

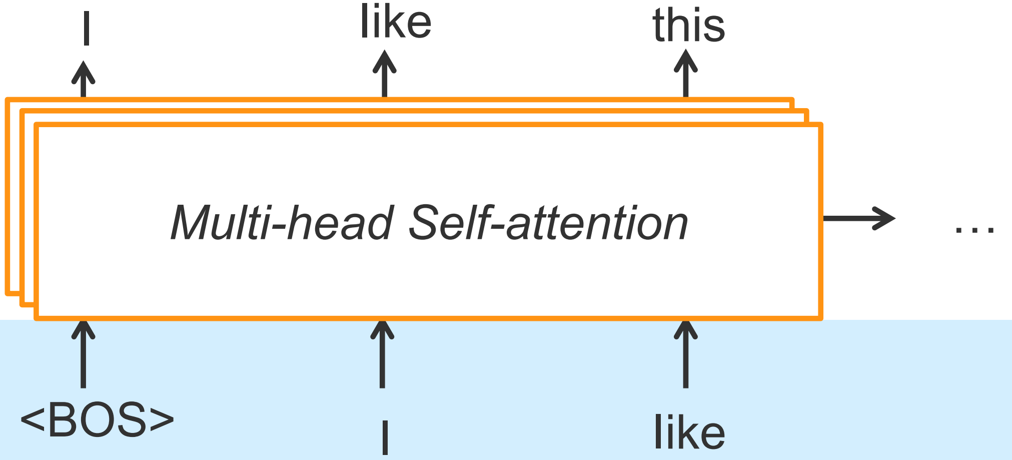
Example:

(I, like, this, …)

$$\cdots p_\theta\,(like \mid I)\, p_\theta(this \mid I, like) \cdots$$

Architecture (1.2)

Transformer



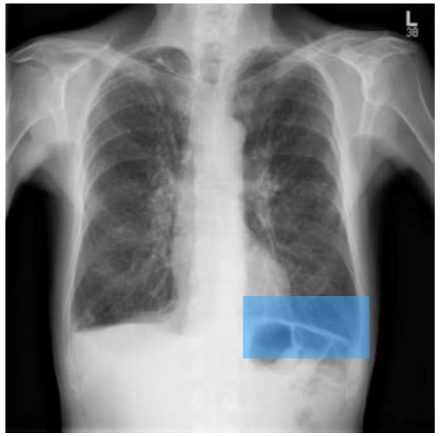
Architecture (2): Conditional Language Model

$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$

- Conditions on additional task-dependent context $\boldsymbol{x}$
  - Machine translation: source sentence

I like this movie. ⟶ Ich mag diesen film.

  - Medical image report generation: medical image



⟶ … There is chronic pleural-parenchymal scarring within the lung bases. No lobar consolidation is seen. …

Architecture (2): Conditional Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Conditions on additional task-dependent context $\boldsymbol{x}$

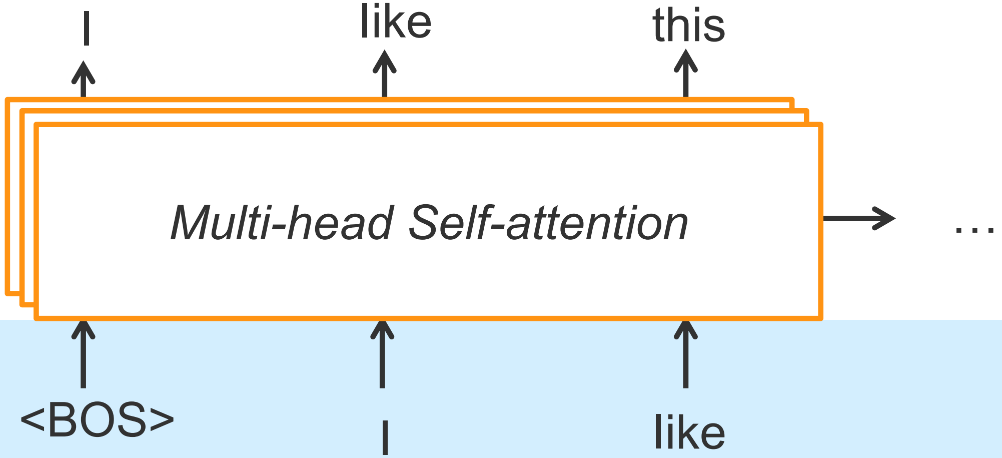
$$p_\theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{x})$$

Architecture (2): Conditional Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Conditions on additional task-dependent context $\boldsymbol{x}$

$$p_\theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{x})$$



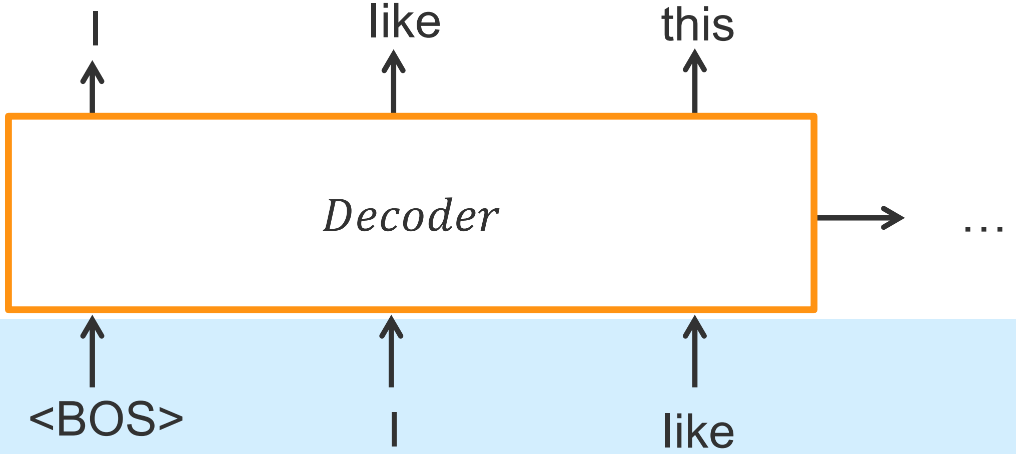
$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Architecture (2): Conditional Language Model

- Conditions on additional task-dependent context $\boldsymbol{x}$

$$p_\theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{x})$$

- Language model as a decoder



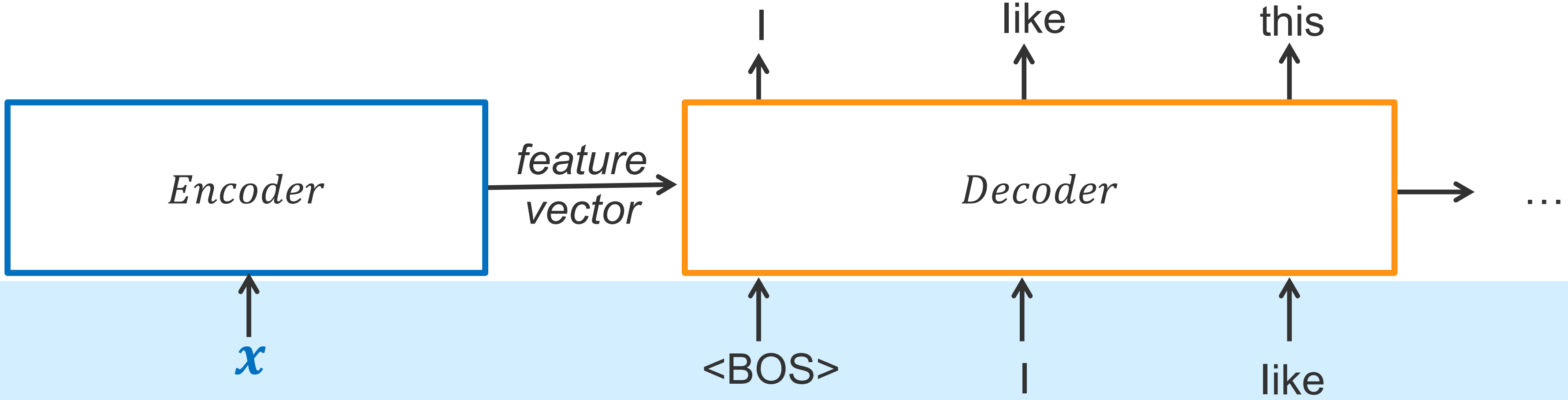
Architecture (2): Conditional Language Model

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Conditions on additional task-dependent context $\boldsymbol{x}$

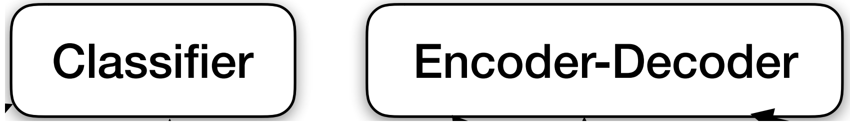
$$p_\theta(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{t=1}^{T} p_\theta(y_t \mid \boldsymbol{y}_{1:t-1}, \boldsymbol{x})$$

- Language model as a decoder
- Encodes context with an encoder



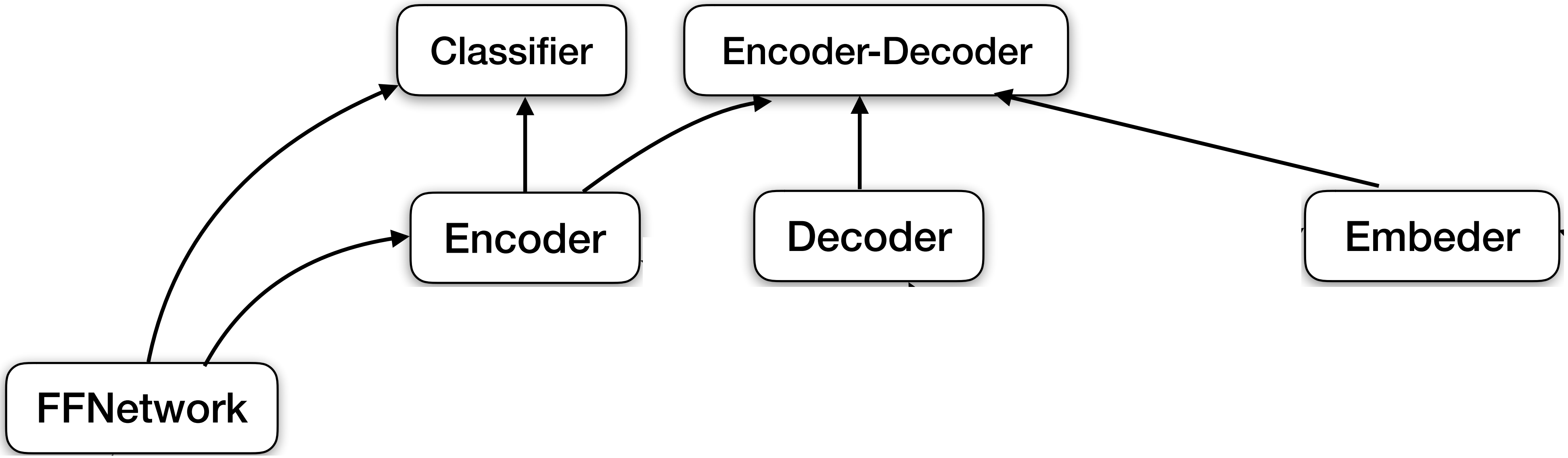
Architecture Graph

$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$



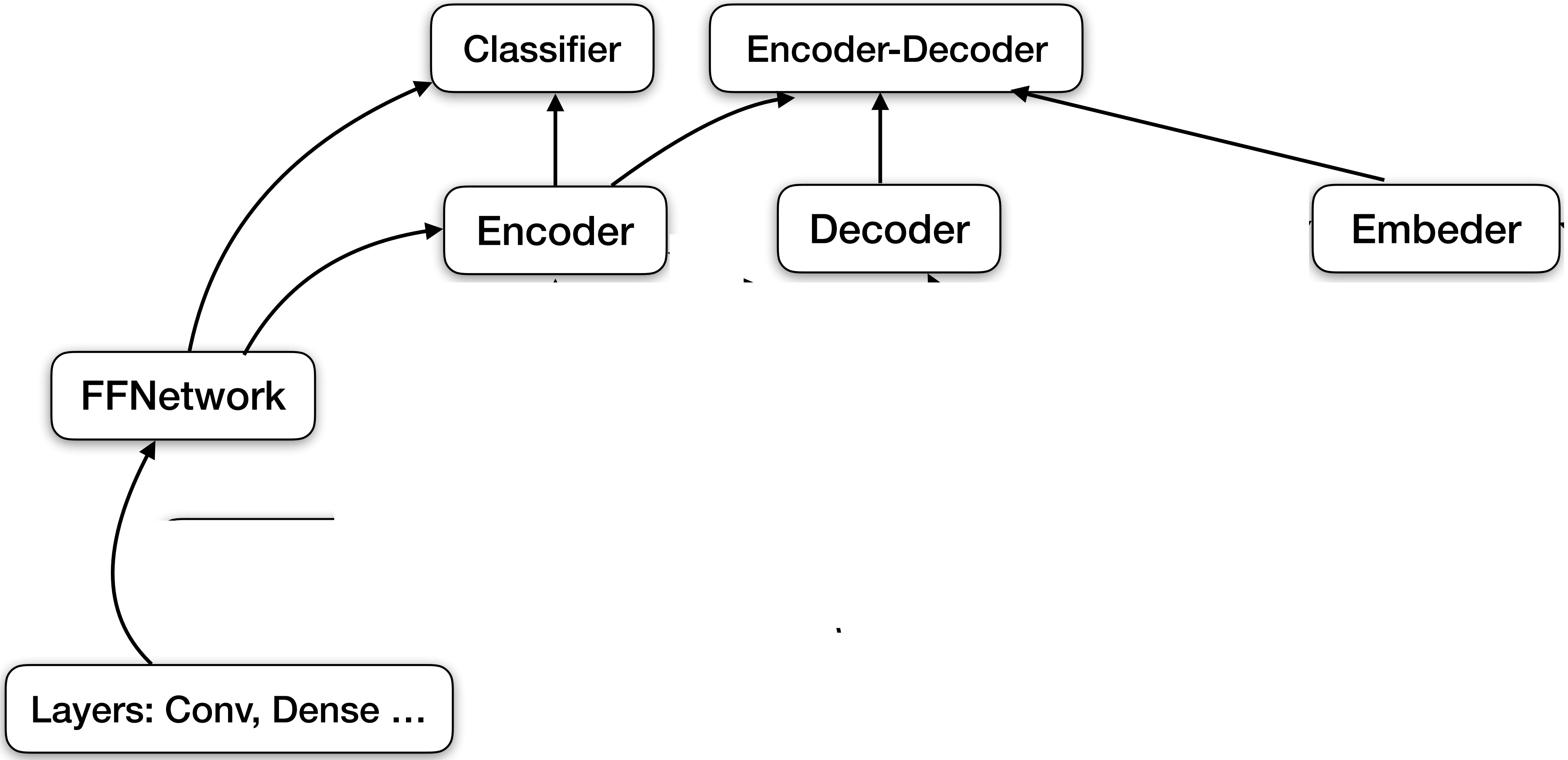
Architecture Graph

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$



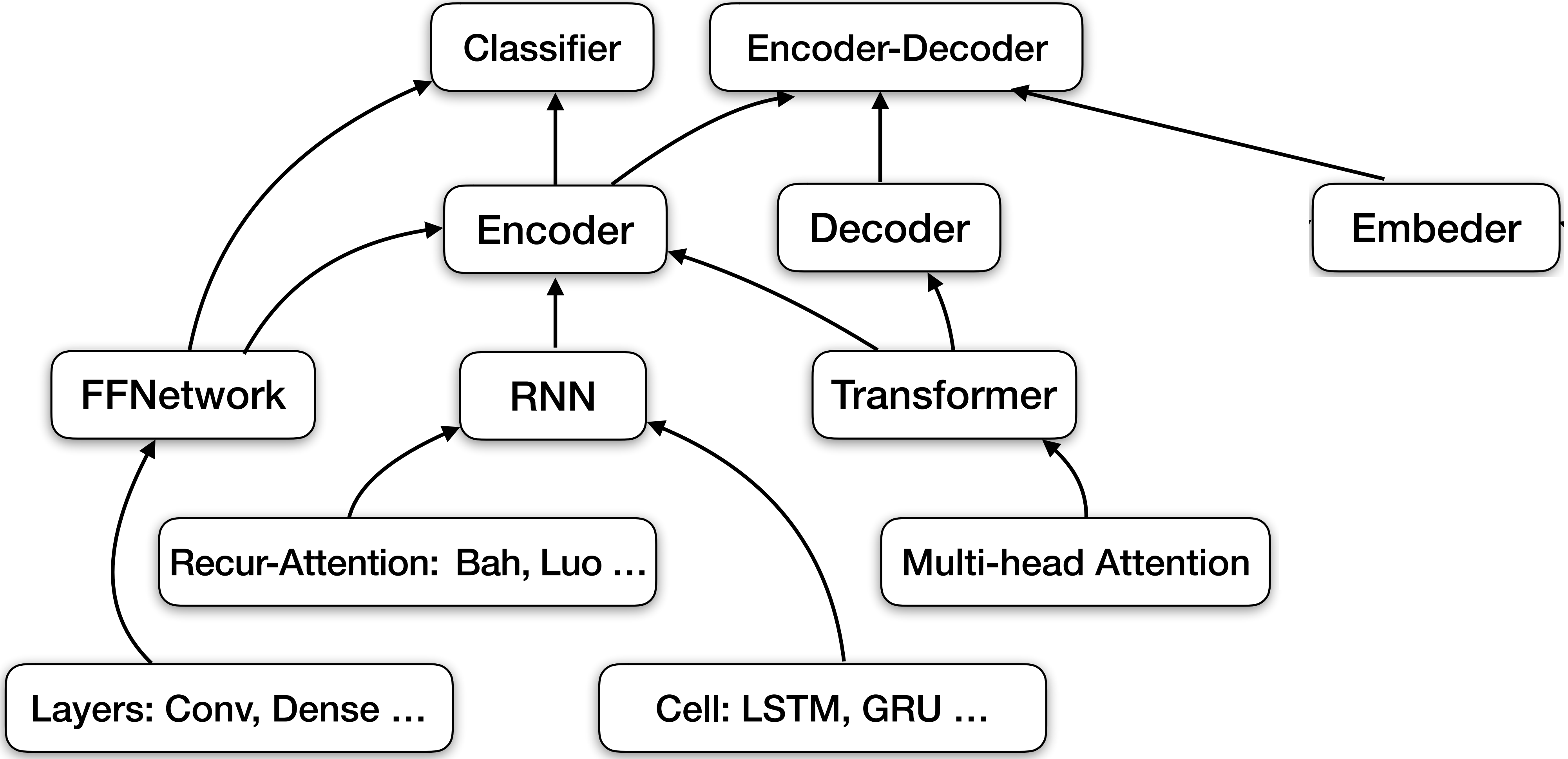
Architecture Graph

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$



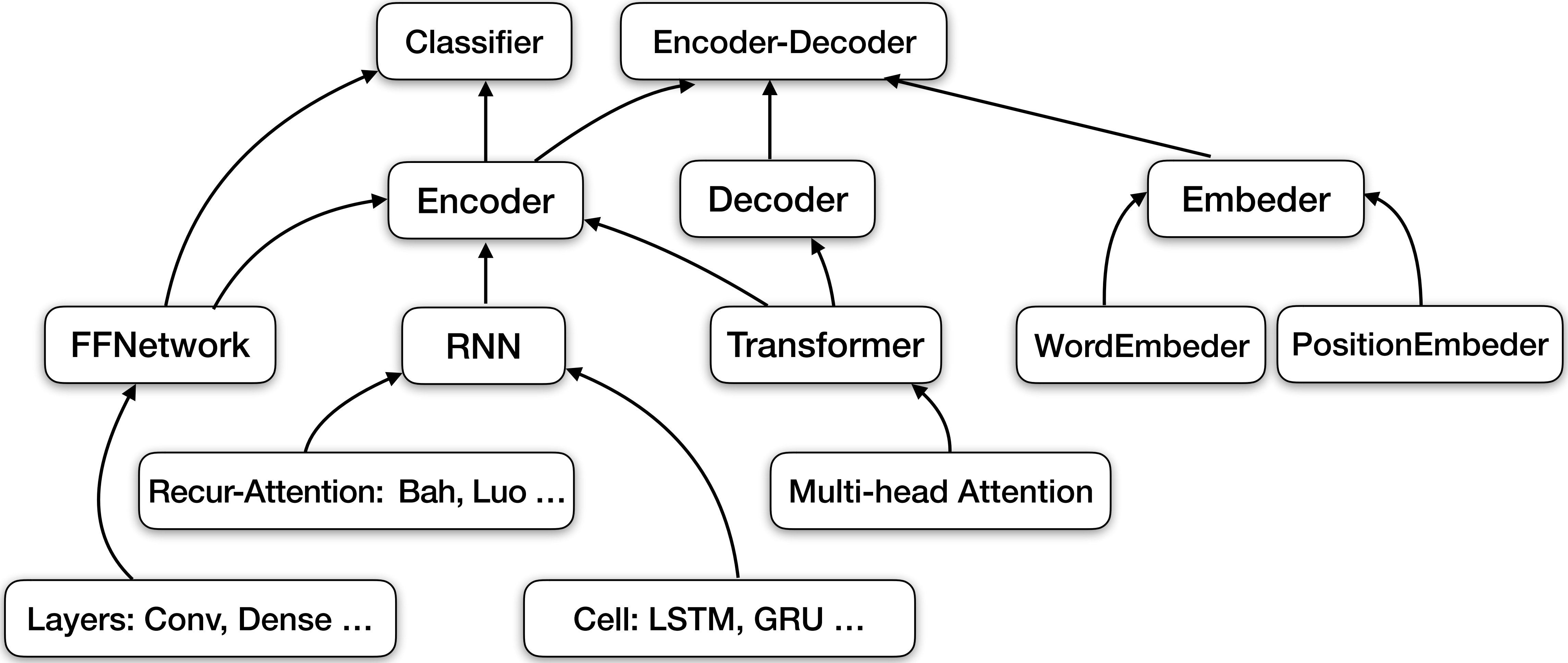
Architecture Graph

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$



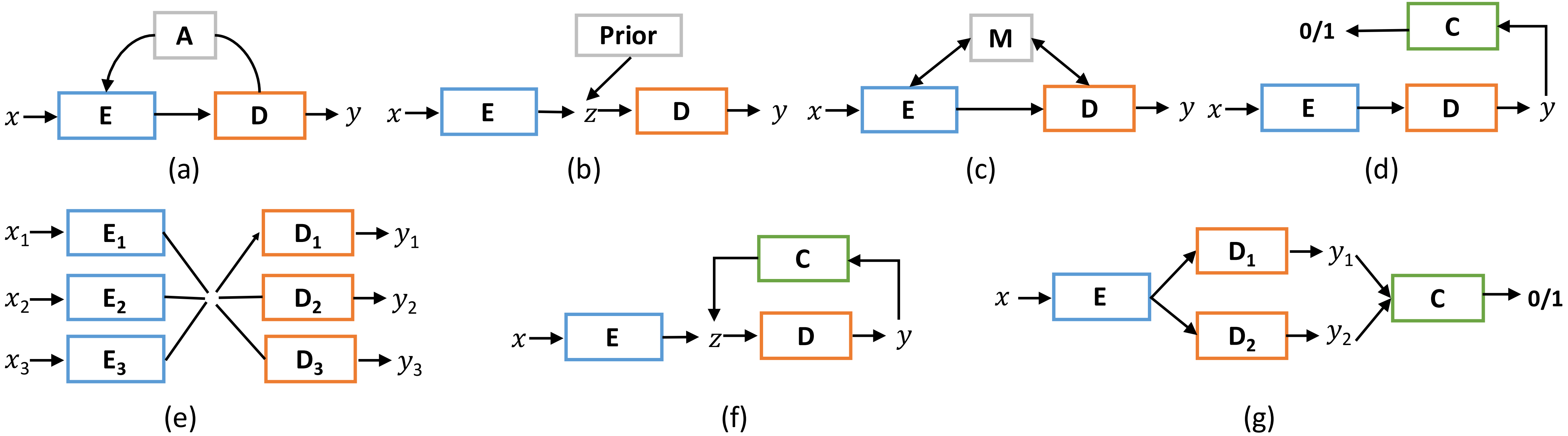
Architecture Graph

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$



Complex Composite Architectures

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$



E refers to encoder, D to decoder, C to Classifier, A to attention, Prior to prior distribution, and M to memory

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint

Loss

Learning

Inference

Architecture

decoder

LSTM RNN

Attention RNN

Transformer

...

encoder

classifier

...

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint

Loss

Learning

Inference

Architecture

decoder

LSTM RNN

Attention RNN

Transformer

...

encoder

classifier

...

Learning, Inference & Loss (1): Maximum Likelihood Estimation

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Given data example $(\boldsymbol{x}^*, \boldsymbol{y}^*)$
- Maximizes log-likelihood of the data

Learning

$$\min_\theta \mathcal{L}_{\text{MLE}} = -\log p_\theta(\boldsymbol{y}^* \mid \boldsymbol{x}^*)$$

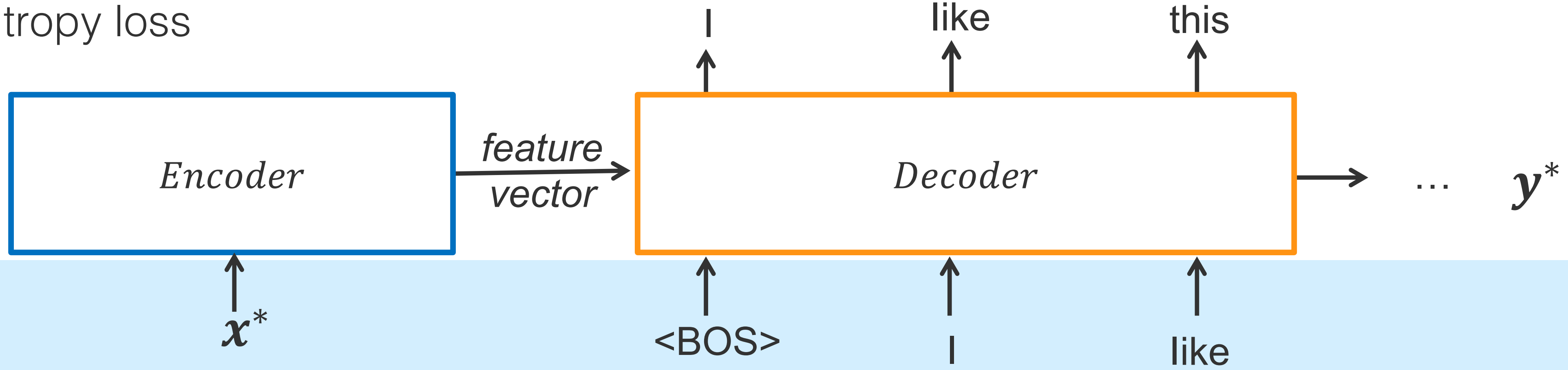
$$= -\prod_{t=1}^{T} p_\theta(y_t^* \mid \boldsymbol{y}_{1:t-1}^*, \boldsymbol{x}^*)$$

Inference

Teacher-forcing decoding:

For every step t, feeds in the previous ground-truth tokens $y_{1:t-1}^*$ to decode next step

Loss Cross-entropy loss



Learning, Inference & Loss (2): Adversarial Learning

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Learning

- A discriminator is trained to distinguish b/w *real data examples* and *fake generated samples*
- The model is trained to fool the discriminator

Loss

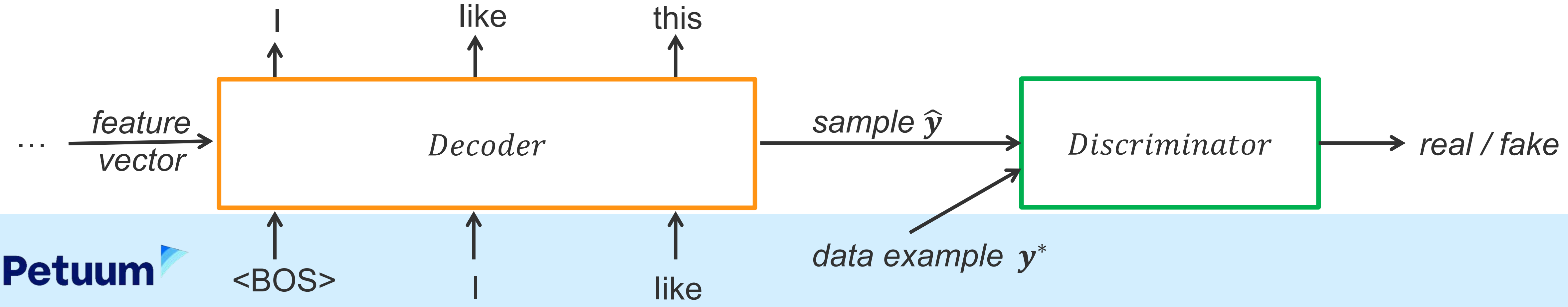
- Binary adversarial loss
- Feature-matching adversarial loss

Inference

Gumbel-softmax decoding:

Uses a differentiable approximation of sample $\hat{\boldsymbol{y}}$ for gradient backpropagation

$$\frac{\partial \mathcal{L}(\hat{\boldsymbol{y}})}{\partial \theta} = \frac{\partial \mathcal{L}(\hat{\boldsymbol{y}})}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \theta}$$



Learning, Inference & Loss (3): Reinforcement Learning

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Learning

- Optimizes *expected* task reward
  - $\mathrm{BLEU}(\hat{\boldsymbol{y}}, \boldsymbol{y}^*)$ for machine translation

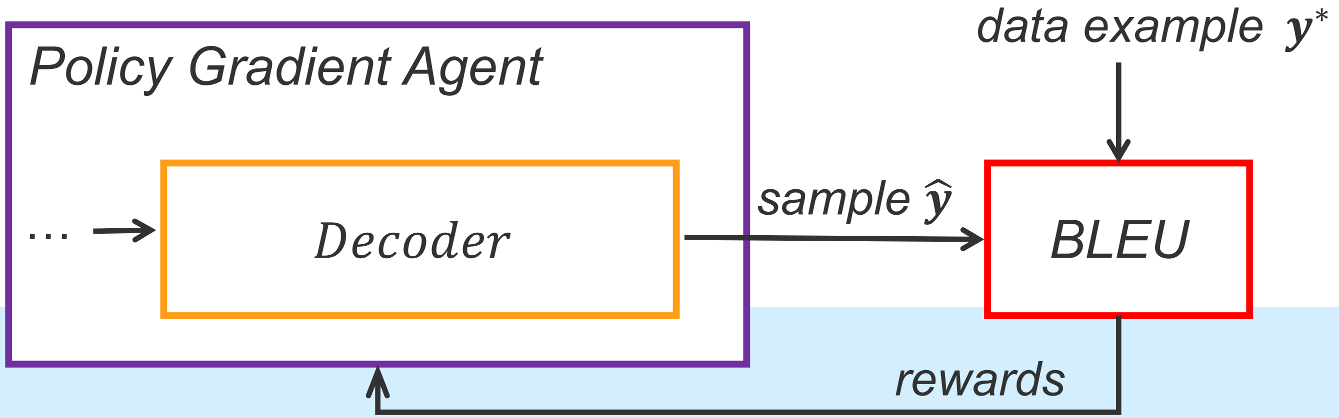
    $\mathbb{E}_{\hat{\boldsymbol{y}} \sim p_\theta(\boldsymbol{y} \mid \boldsymbol{x})}[\ \mathrm{BLEU}(\hat{\boldsymbol{y}}, \boldsymbol{y}^*)\ ]$

Loss

- Policy gradient loss
- Policy gradient loss w/ baseline
- …

Inference

- Greedy decoding
- Sampling decoding
- Beam search decoding
- Top-*k* / Top-*p* decoding
- …



ML Components

$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint	Loss	Learning	Inference	Architecture
	Cross-entropy	MLE	Teacher-forcing	decoder
	Binary Adv loss	Adversarial	Gumbel-softmax	LSTM RNN
	Matching Adv loss	Reinforcement	Sample	Attention RNN
	PG loss	Adv + RL	Greedy	Transformer
	PG loss + baseline	Structured super.	Beam-search	…
	…	Reward-aug.	Top-k sample	encoder
		…	…	classifier
				…

ML Components

$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint	Loss	Learning	Inference	Architecture
	Cross-entropy	MLE	Teacher-forcing	decoder
	Binary Adv loss	Adversarial	Gumbel-softmax	LSTM RNN
	Matching Adv loss	Reinforcement	Sample	Attention RNN
	PG loss	Adv + RL	Greedy	Transformer
	PG loss + baseline	Structured super.	Beam-search	…
	…	Reward-aug.	Top-k sample	encoder
		…	…	classifier
				…

Petuum

Constraint (1): Conventional Constraints

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Many choices for get different statistical properties:
  - Normality, Sparsity, KL, sum, …

Constraint (2): Structured Knowledge

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Structured knowledge as constraints

Sentiment classification:

- ``Food was good, but the service was very disappointing.''

Logic rule: $f(\boldsymbol{x} = \text{sentence}, \boldsymbol{y} = \text{sentiment}) = \text{truth value}$

- Sentence $\boldsymbol{x}$ with structure A-but-B => sentiment of B dominates

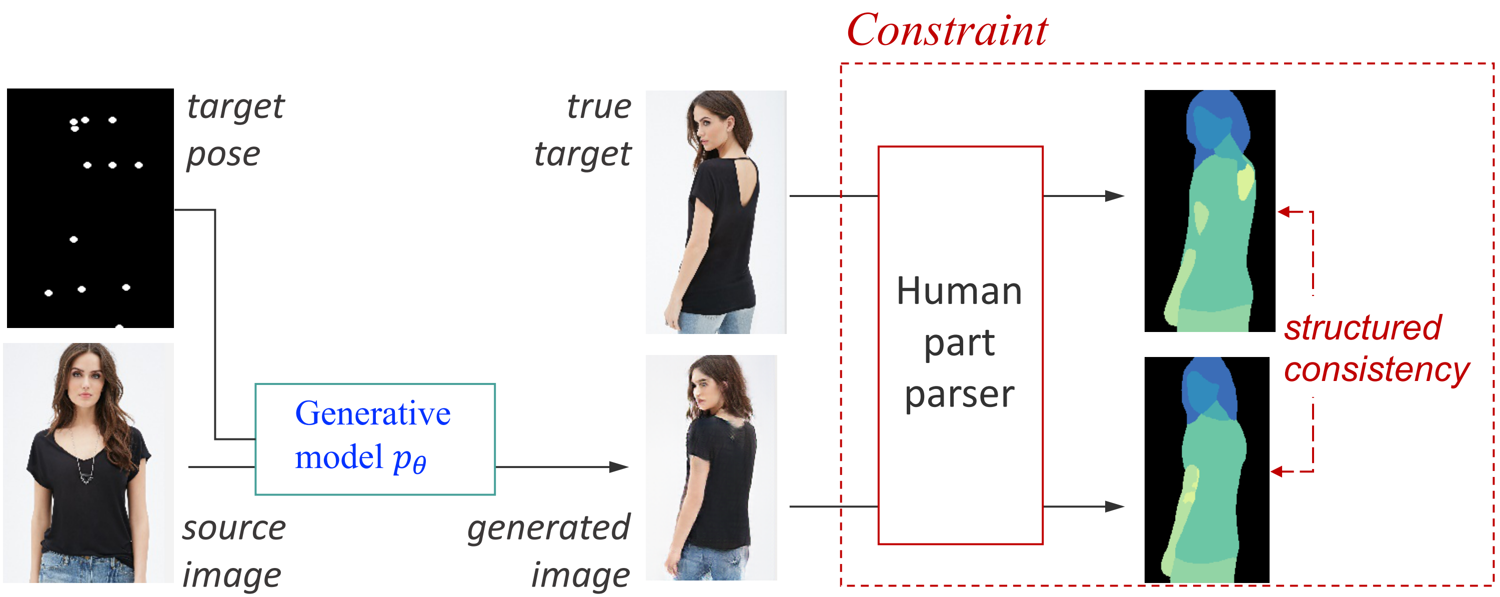
Constraint (2): Structured Knowledge

$$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$$

- Structured knowledge as constraints

Human Image Generation

$$f(\boldsymbol{y} = \text{generated}, \boldsymbol{o} = \text{ground truth}) = \text{match score}$$



Constraint (2): Structured Knowledge

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

- Structured knowledge as constraints
  - Constraint function: $f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{o}) \in \mathbb{R}$
  - Model: $p_\theta(\boldsymbol{y}|\boldsymbol{x})$
  - Variational Knowledge Regularization [Hu et al., 2016, 2018; Ga

Variational distribution

$$\mathcal{L}_{VKR}(q, \boldsymbol{\theta}) = \mathbb{E}_{q(\boldsymbol{y}|\boldsymbol{x})}[\, f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{o})] - \mathrm{KL}\big(q(\boldsymbol{y}|\boldsymbol{x}) \,||\, p_\theta(\boldsymbol{y}|\boldsymbol{x})\big)$$

- Learning procedure: at iteration n

$$q^{n+1}(\boldsymbol{y}|\boldsymbol{x}) \propto p_{\theta^n}(\boldsymbol{y}|\boldsymbol{x}) \exp\{f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{o})\}$$

Combines the model and the knowledge
-- teacher model

$$\boldsymbol{\theta}^{n+1} = \mathrm{argmax}_\theta \mathbb{E}_{q^{n+1}(\boldsymbol{y}|\boldsymbol{x})}[\, \log p_\theta(\boldsymbol{y}|\boldsymbol{x}) \,]$$

The model imitates the teacher model predictions
-- student model

Petuum

ML Components

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint	Loss	Learning	Inference	Architecture
L1 / L2	Cross-entropy	MLE	Teacher-forcing	decoder
Logical	Binary Adv loss	Adversarial	Gumbel-softmax	LSTM RNN
Structured	Matching Adv loss	Reinforcement	Sample	Attention RNN
…	PG loss	Adv + RL	Greedy	Transformer
	PG loss + baseline	Structured super.	Beam-search	…
	…	Reward-aug.	Top-k sample	encoder
			…	classifier
				…

Holistic View

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint	Loss	Learning	Inference	Architecture
				decoder
L1 / L2	Cross-entropy	MLE	Teacher-forcing	LSTM RNN
Logical	Binary Adv loss	Adversarial	Gumbel-softmax	Attention RNN
Structured	Matching Adv loss	Reinforcement	Sample	Transformer
…	PG loss	Adv + RL	Greedy	…
	PG loss + baseline	Structured super.	Beam-search	
	…	Reward-aug.	Top-k sample	encoder
			…	classifier
				…

Holistic View

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint	Loss	Learning	Inference	Architecture
L1 / L2	Cross-entropy	MLE	Teacher-forcing	decoder
Logical	Binary Adv loss	Adversarial	Gumbel-softmax	LSTM RNN
Structured	Matching Adv loss	Reinforcement	Sample	Attention RNN
…	PG loss	Adv + RL	Greedy	Transformer
	PG loss + baseline	Structured super.	Beam-search	…
	…	Reward-aug.	Top-k sample	encoder
		…	…	classifier
				…

Holistic View

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

Constraint	Loss	Learning	Inference	Architecture
L1 / L2	Cross-entropy	MLE	Teacher-forcing	decoder
Logical	Binary Adv loss	Adversarial	Gumbel-softmax	LSTM RNN
Structured	Matching Adv loss	Reinforcement	Sample	Attention RNN
…	PG loss	Adv + RL	Greedy	Transformer
	PG loss + baseline	Structured super.	Beam-search	…
	…	Reward-aug.	Top-k sample	encoder
		…	…	classifier
				…

Holistic View

$\min_\theta \mathcal{L}(\theta, D) + \Omega(\theta)$

called as subroutines

uniform interfaces

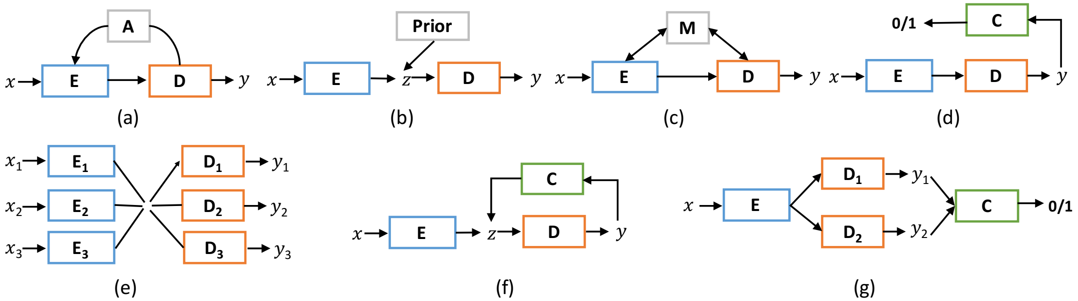
Constraint	Loss	Learning	Inference	Architecture
L1 / L2	Cross-entropy	MLE	Teacher-forcing	decoder
Logical	Binary Adv loss	Adversarial	Gumbel-softmax	LSTM RNN
Structured	Matching Adv loss	Reinforcement	Sample	Attention RNN
…	PG loss	Adv + RL	Greedy	Transformer
	PG loss + baseline	Structured super.	Beam-search	…
	…	Reward-aug.	Top-k sample	encoder
		…	…	classifier
				…

Composable ML – Take-Home Message

- Composable ML
  - Basic "musical notes" for complex ML systems
  - Structure/rules for combining notes into "chords"/"rhythms" …
  - … and chords/rhythms into compositions
  - (or just think of it as Lego for ML)

- **Today** – Symbolic programming via Petuum Texar open source

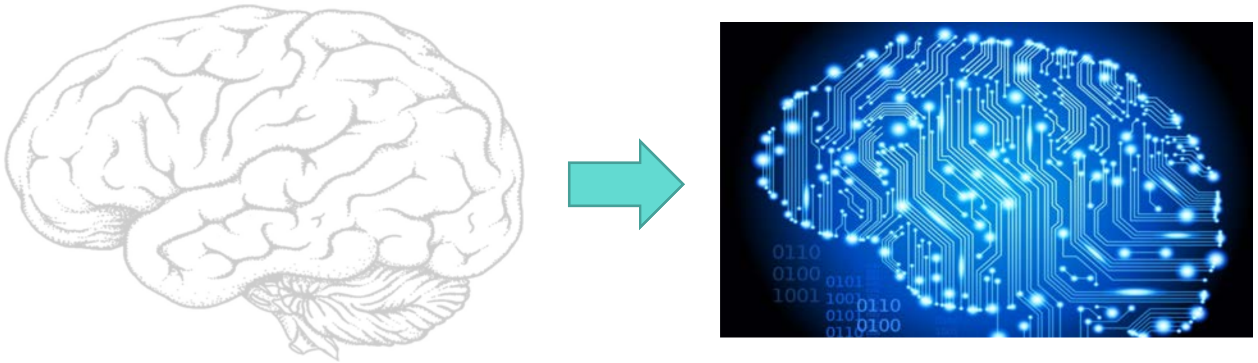
- **Future** – Drag and Play graphical UI



PETUUM

Texar

Compose your ML applications like playing building blocks



Decomposing Machine Learning

$$\min_{\theta} \mathcal{L}(\theta, D) + \Omega(\theta)$$

constraint

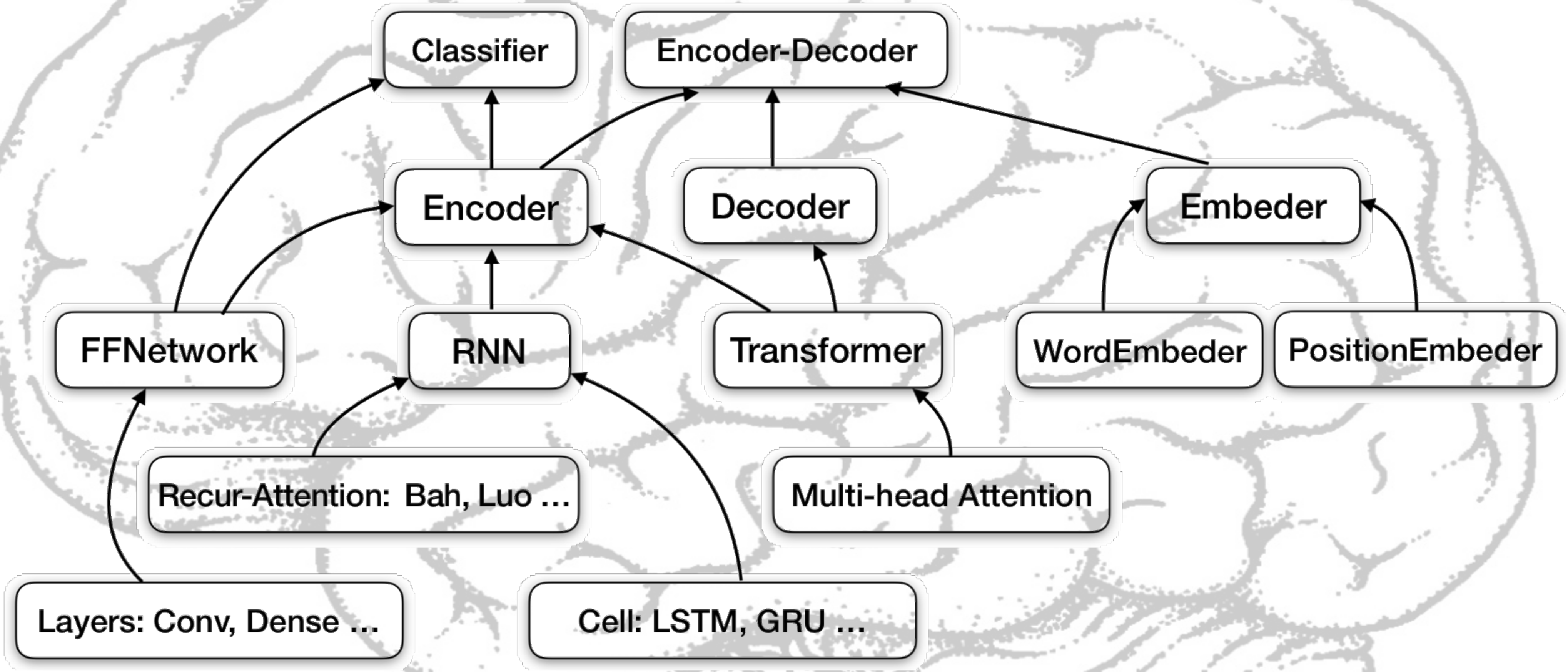
learning procedure

experience (data)

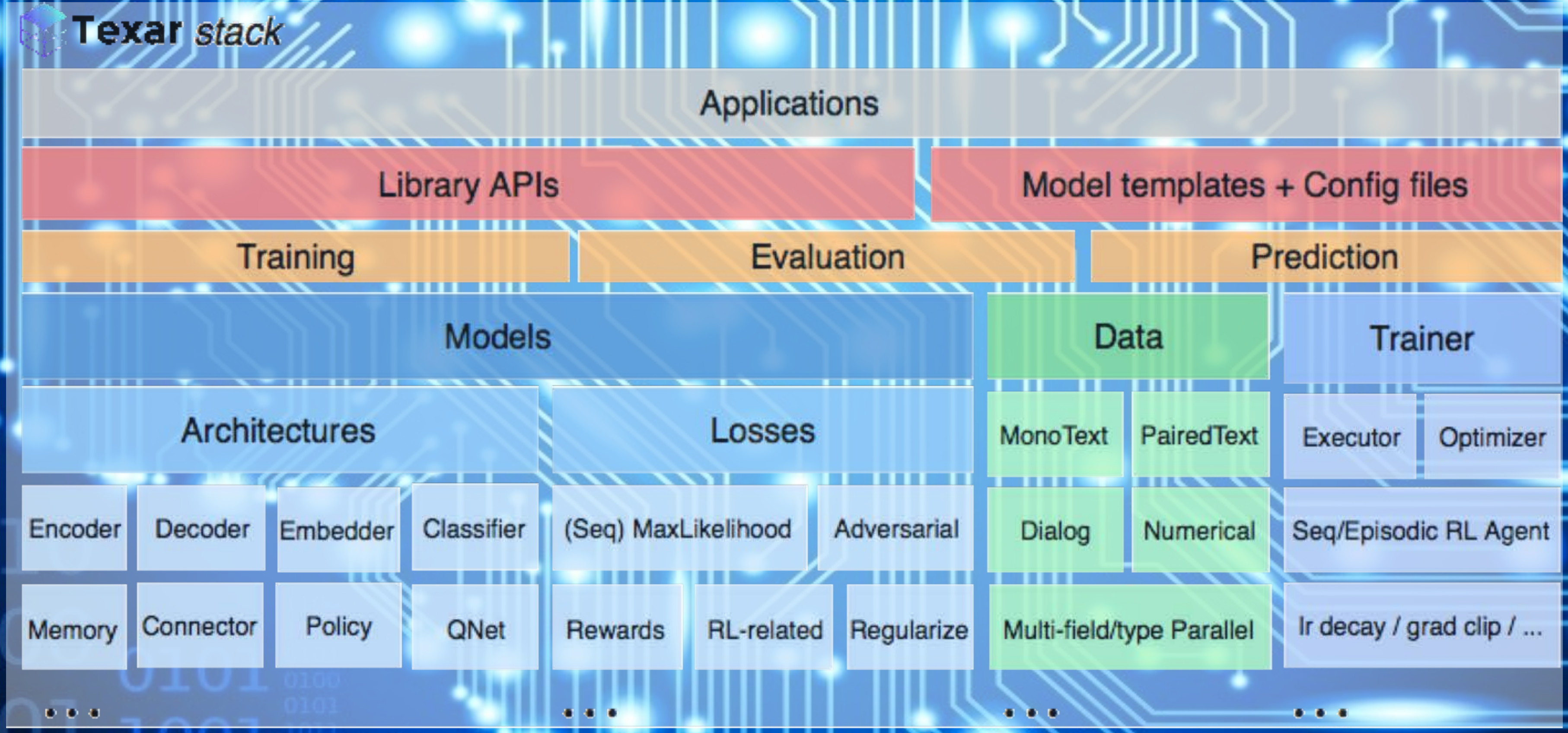
loss

model architecture/
inference procedure

Expert's Intellectual "View" of Composable ML



Texar Stack – Operationalized "View" of Composable ML



Module Catalog in Texar

Model architecture

- **Encoder**
  - UnidirectionalRNNEncoder
  - BidirectionalRNNEncoder
  - HierarchicalRNNEncoder
  - ConvEncoder
  - TransformerEncoder
  - …
- **Decoder**
  - BasicRNNDecoder
  - AttentionRNNDecoder
    - BahdananAttn
    - LuongAttn
    - MonotonicAttn
  - TransformerDecoder
  - Greedy/Sample/BeamSearch/ GumbelSoftmax/… Decoding
  - …
- **Embedder**
  - WordEmbedder (one-hot / soft)
  - PositionEmbedder
    - Parametrized
    - Sinusoids
  - …
- **Classifier/ Discriminator**
  - RNNClassifier
  - ConvClassifier
  - HierarchicalClassifier
  - …
- **Connector**
  - MLPTransformer
  - Stochastic
  - ReparameterizedStochastic
  - Concat
  - Forward
  - …
- . . .

Model loss

- **Loss**
  - MLE Loss
    - (Sequence) Cross-entropy
    - …
  - Adversarial Loss
    - Binary Adversarial Loss
    - …
  - Rewards
  - …

Trainer

- **RL Agent**
  - Seq RL Agent
    - Seq Policy Gradient Agent
    - …
  - Episodic RL Agent
    - Policy Gradient Agent
    - DQN Agent
    - Actor-critic Agent
    - …
- **Optimization**
  - Optimizer
    - Adam/SGD/…
  - Learning Rate Decay
    - Piecewise/Exp/…
  - …

Data

- **Data**
  - MonoText
  - PairedText
  - MultiAligned
  - Dialog
  - Numerical
  - …
- . . .

Texar Highlights

Modularized

Assembles any complex model like playing building blocks

Versatile

Supports a large variety of models/algorithms/ applications …

Extensible

Allows to plug in any customized or external modules

Petuum

Running Example: Machine Translation

raw data

cleaning
tokenizing
vocabulary
truncation
…

source.dat

I like this movie.

Lovely and poignant

Insanely hilarious!

…

target.dat

Ich mag diesen film.

Schön und ergreifend

Wahnsinnig witzig!

…

clean data

model

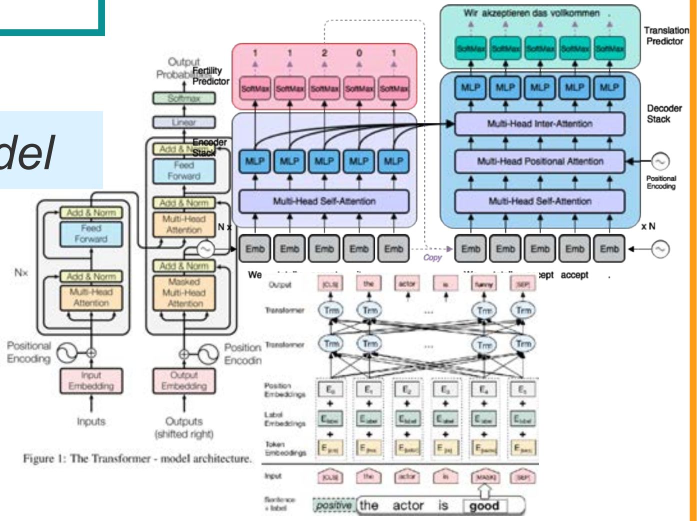


Figure 1: The Transformer - model architecture.

training

Maximum likelihood training

Reinforcement learning

Adversarial

Finetuning

evaluation
post-processing

…

Running Example: Machine Translation - Data Preparation

- Input: a source sentence:

  I like this movie.

- Output: a target sentence:

  Ich mag diesen film.

- Dataset:

source.txt

I like this movie.
Lovely and poignant
Insanely hilarious!
…

target.txt

Ich mag diesen film.
Schön und ergreifend
Wahnsinnig witzig!
…

vocab.txt

I
like
this
movie
Ich
mag

Running Example: Machine Translation - Programming

Running Example: Machine Translation - Programming

Data

```python
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
```

```
{
    'batch_size': 64,
    'num_epochs': 10,
    'shuffle': True,
    'source_dataset': {
        'files': 'source.txt',
        'vocab_file': 'vocabulary.txt'
        'max_seq_length': 100,
        'bos_token': '<BOS>',
        'eos_token': '<EOS>',
        'embedding_init': { … }
    },
    'target_dataset': {
        'files': 'target.txt',
    },
    # …
}
```

Running Example: Machine Translation - Programming

Data

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
```

Architecture
& Inference

Running Example: Machine Translation - Programming

Data

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
```

Architecture & Inference

```
{
  embedding_dim
  dropout
  initialization
  …
}
```

Running Example: Machine Translation - Programming

Data

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
```

Architecture & Inference

```
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
```

```
{
  #blocks
  #heads
  hidden dim
  output dim
  dropout
  initialization
}
```

Running Example: Machine Translation - Programming

Data

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])
```

Architecture & Inference

Running Example: Machine Translation - Programming

Data

Architecture & Inference

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])
# Build decoder
decoder = AttentionRNNDecoder(memory=enc_outputs,
                              hparams=decoder_hparams)
# Maximum Likelihood Estimation
## Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy=’teacher-forcing’,
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
```

Running Example: Machine Translation - Programming

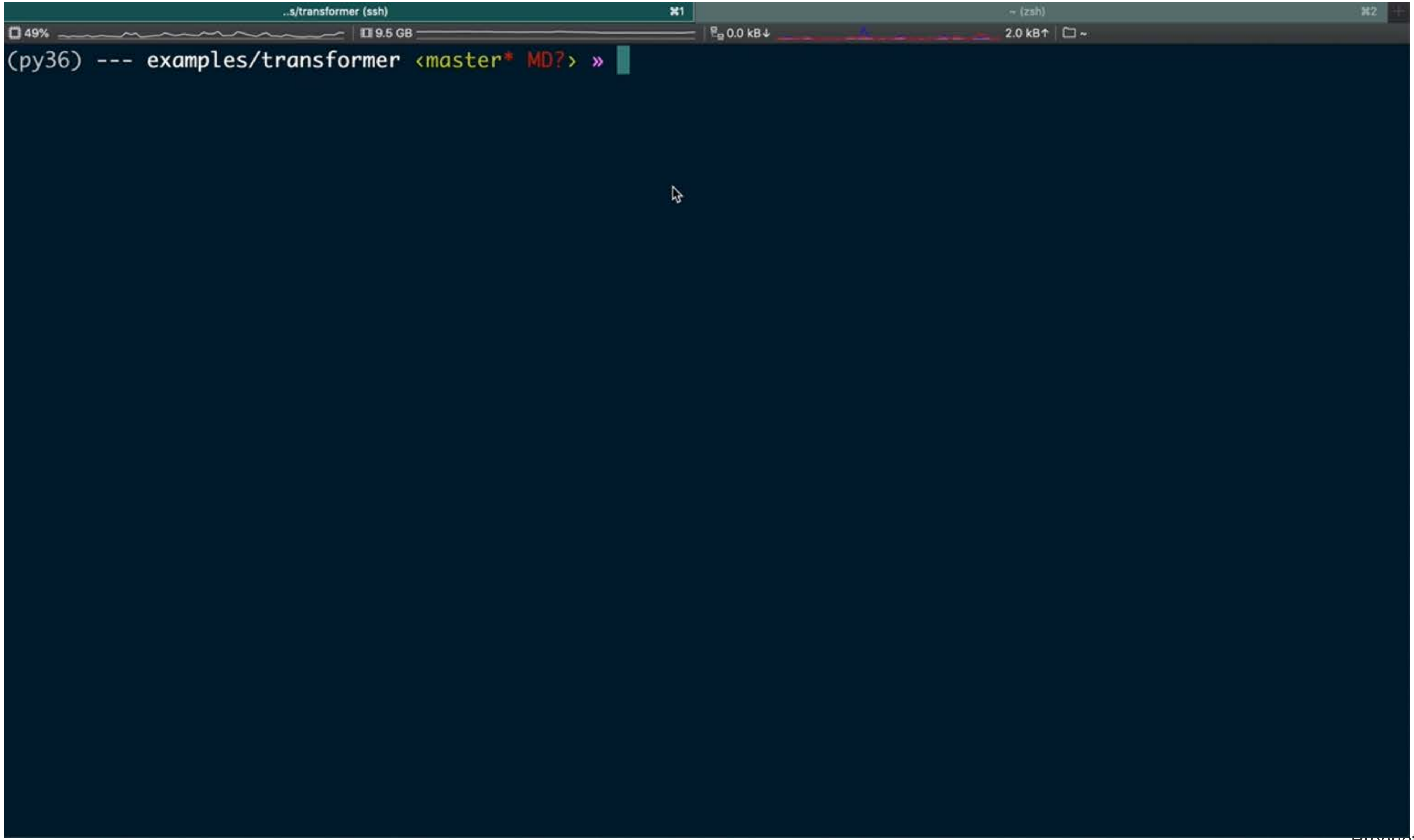
Data

Architecture & Inference

Learning Loss

```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                                    batch['source_length'])
# Build decoder
decoder = AttentionRNNDecoder(memory=enc_outputs,
                                             hparams=decoder_hparams)
# Maximum Likelihood Estimation
## Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy=’teacher-forcing’,
                                        inputs=embedder(batch['target_text_ids']),
                                        seq_length=batch['target_length']-1)
## Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
   labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

DEMO



..s/transformer (ssh)
~ (zsh)
49%
9.5 GB
0.0 kB
2.0 kB
(py36) --- examples/transformer <master* MD?> »

Example (cont'd): Machine Translation

Data

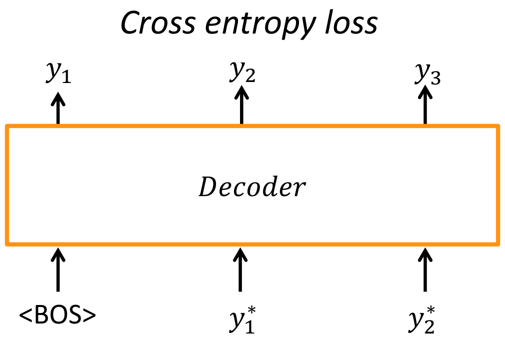
Architecture & Inference

Learning Loss

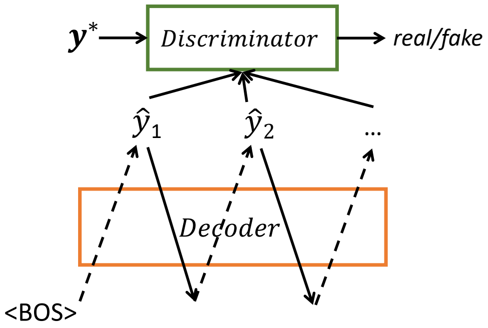
```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                                    batch['source_length'])
# Build decoder
decoder = AttentionRNNDecoder(memory=enc_outputs,
                                                    hparams=decoder_hparams)
# Maximum Likelihood Estimation
## Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                                            inputs=embedder(batch['target_text_ids']),
                                            seq_length=batch['target_length']-1)
## Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
   labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

Maximum likelihood Estimation

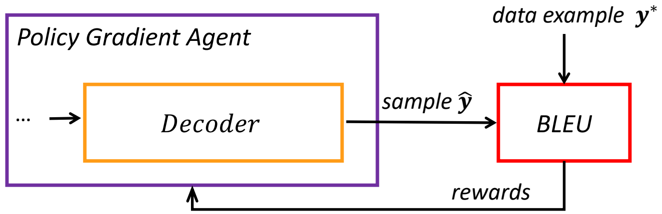
Different Learning Algorithms



- Maximum Likelihood Estimation



- Adversarial Learning



- Reinforcement Learning

Petuum

Switching between Learning Algorithms

Data

Architecture & Inference

Learning Loss

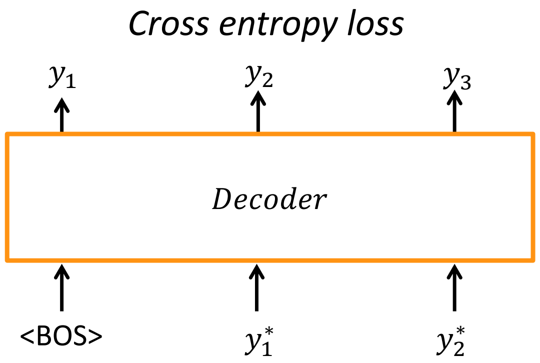
```
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])
# Build decoder
decoder = AttentionRNNDecoder(memory=enc_outputs,
                              hparams=decoder_hparams)
# Maximum Likelihood Estimation
## Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
## Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

Keep unchanged

Maximum likelihood Estimation

Switching from MLE to Adversarial Learning

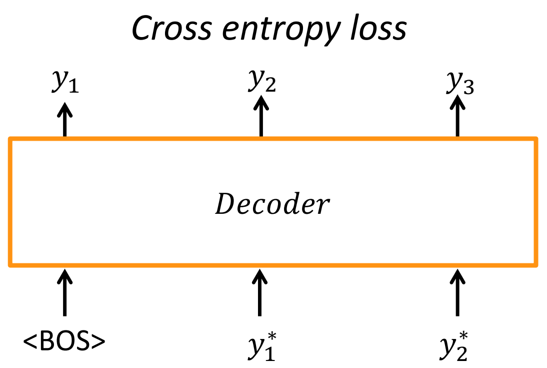
- Maximum likelihood




```
# Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
# Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

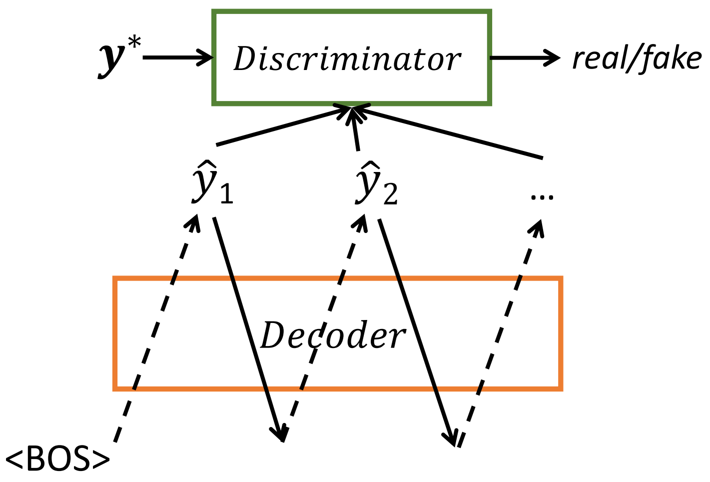
Switching from MLE to Adversarial Learning

- Maximum likelihood




```
# Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
# Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

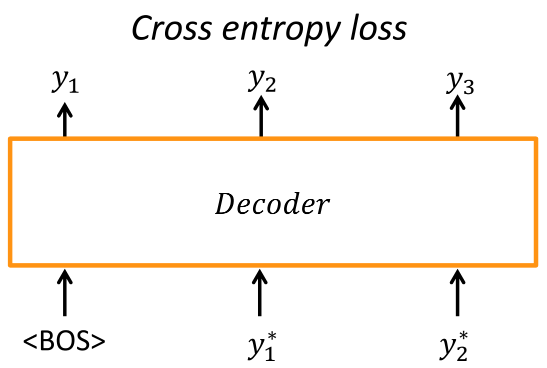
- Adversarial learning




```
# Gumbel-softmax decoding
helper = GumbelSoftmaxTrainingHelper(
    start_tokens=[BOS]*batch_size, end_token=EOS, embedding=embedder)
outputs, _, _ = decoder(helper=helper)

discriminator = Conv1DClassifier(hparams=conv_hparams)
# Binary adversarial loss
G_loss, D_loss = binary_adversarial_losses(
    embedder(batch['target_text_ids'][:, 1:]),
    embedder(soft_ids=softmax(outputs.logits)),
    discriminator)
```

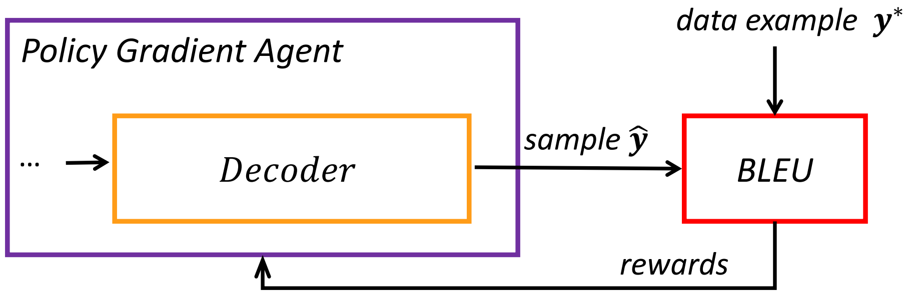
Switching from MLE to Reinforcement Learning

- Maximum likelihood




```
# Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
# Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

- Reinforcement learning



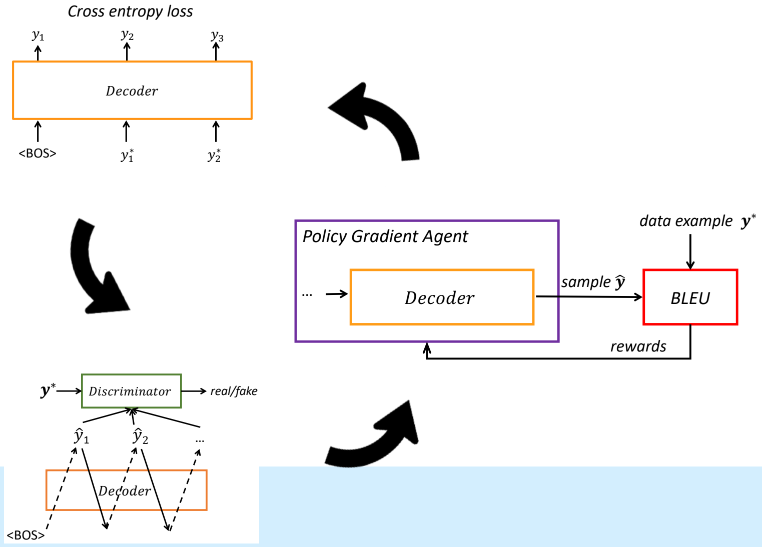

```
# Random sample decoding
outputs, length, _ = decoder(decoding_strategy='random_sample',
                             start_tokens=[BOS]*batch_size, end_token=EOS,
                             embedding=embedder)
# Policy gradient agent for learning
agent = SeqPGAgent(
    samples=outputs.sample_id, logits=outputs.logits, seq_length=length)

for _ in range(STEPS):
    samples = agent.get_samples()
    rewards = BLEU(batch['target_text_ids'], samples)  # Reward
    agent.observe(rewards)
```

Summary of MT in Texar

- Highly modularized programming
  - Data, architecture, loss, inference, learning, …
  - Intuitive conceptual-level APIs

```python
# Read data
dataset = PairedTextData(data_hparams)
batch = DataIterator(dataset).get_next()
# Encode
embedder = WordEmbedder(dataset.vocab.size, hparams=embedder_hparams)
encoder = TransformerEncoder(hparams=encoder_hparams)
enc_outputs = encoder(embedder(batch['source_text_ids']),
                      batch['source_length'])
# Build decoder
decoder = AttentionRNNDecoder(memory=enc_outputs,
                              hparams=decoder_hparams)
# Maximum Likelihood Estimation
## Teacher-forcing decoding
outputs, length, _ = decoder(decoding_strategy='teacher-forcing',
                             inputs=embedder(batch['target_text_ids']),
                             seq_length=batch['target_length']-1)
## Cross-entropy loss
loss = sequence_sparse_softmax_cross_entropy(
    labels=batch['target_text_ids'][:,1:], logits=outputs.logits, seq_length=length)
```

- Easy switch between learning algorithms
  - Plug in & out modules
  - No changes to irrelevant parts

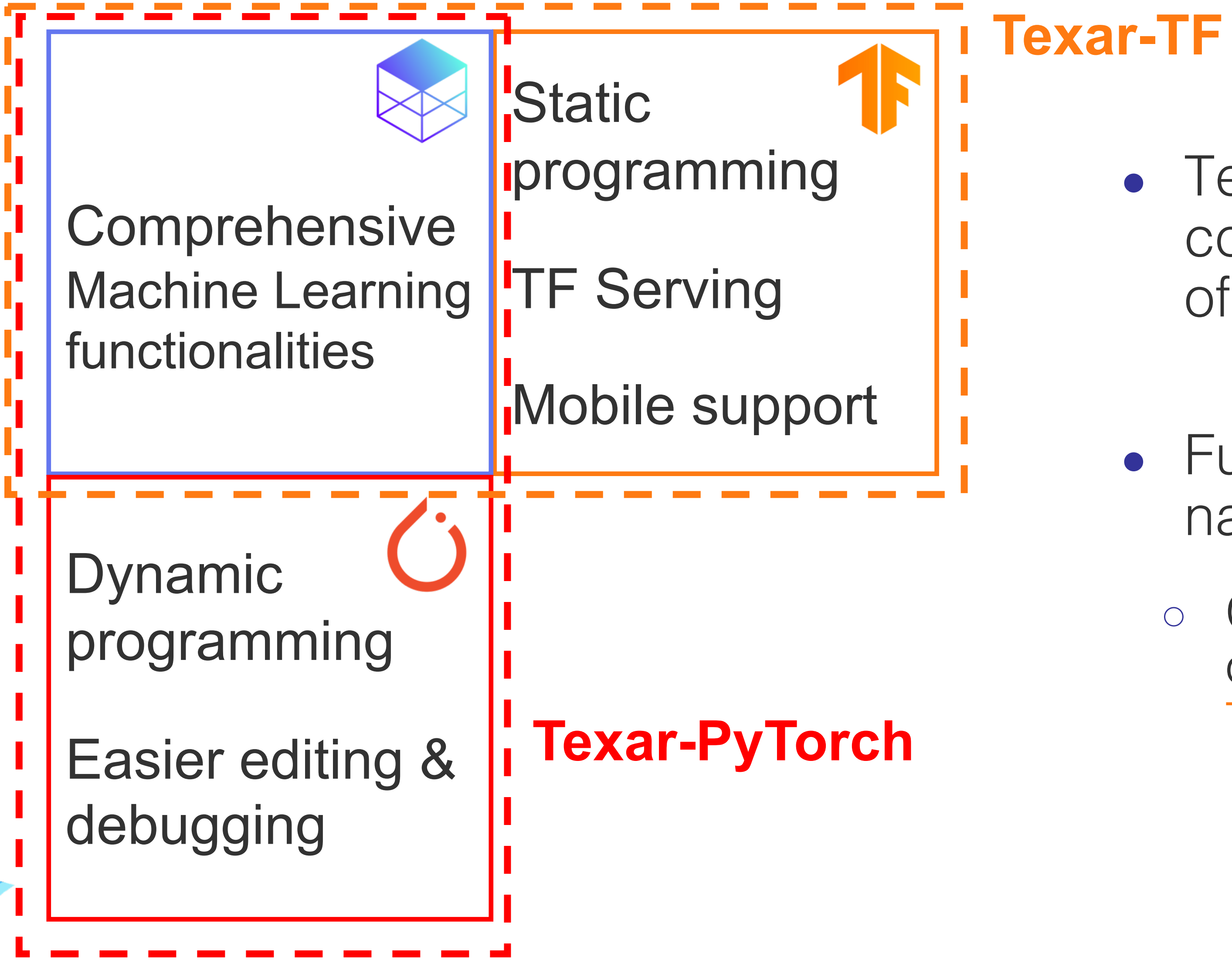


Other Features (I): Support of TensorFlow and PyTorch

- Texar is built upon TF and PyTorch
  - Texar-TF & Texar-PyTorch: mostly the same interfaces!
  - Higher-level intuitive APIs without loss of flexibility
  - Lots of ML components ready to use
- Combine the best design of TF and PyTorch
  - TF:
    - Easy and efficient data processing APIs
    - Excellent factorization of ML modules
    - Turnkey model training processor
  - PyTorch:
    - Intuitive programming interfaces
    - Transparent variable scope and sharing to users

Texar

Other Features (I): Support of TensorFlow and PyTorch

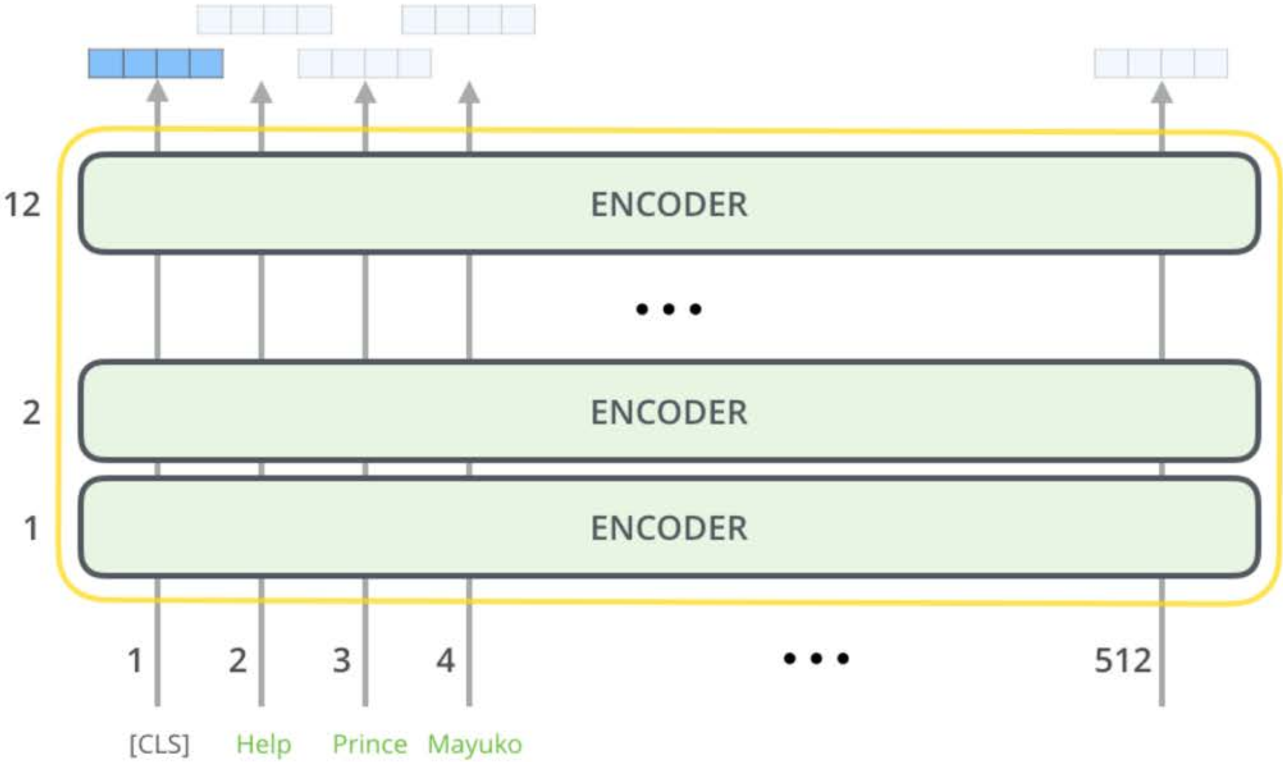


- Texar provides the same comprehensive supports of ML functionalities
- Fully compatible with native TF & PyTorch APIs
  - Get all other useful features of TF or PyTorch with Texar-TF & Texar-PyTorch

Other Features (II): SOTA Pretrained Models

Bert [Devlin et al., 2018]

contextual embedding



input sentence

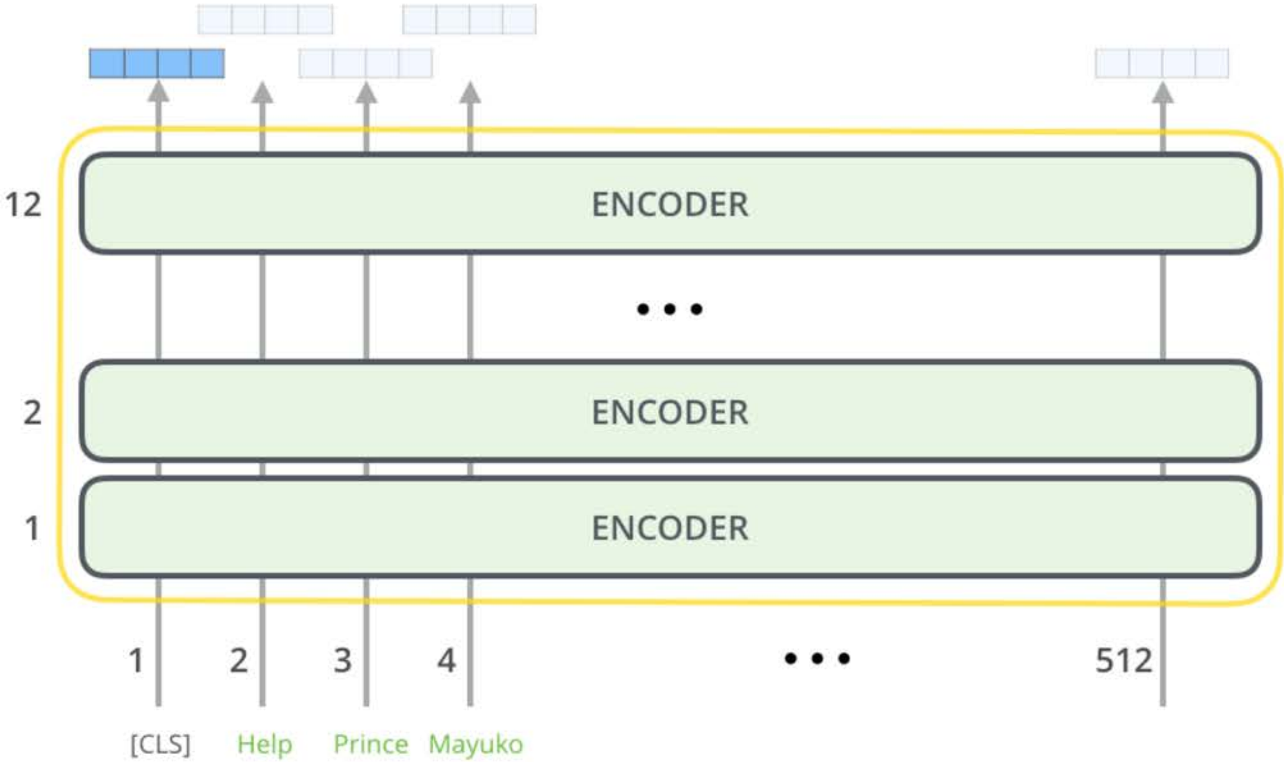
Other Features (II): SOTA Pretrained Models

Bert [Devlin et al., 2018]

feature extraction

```
model = BertEncoder()
features = model(input_ids, input_length, segment_ids)
```

contextual embedding

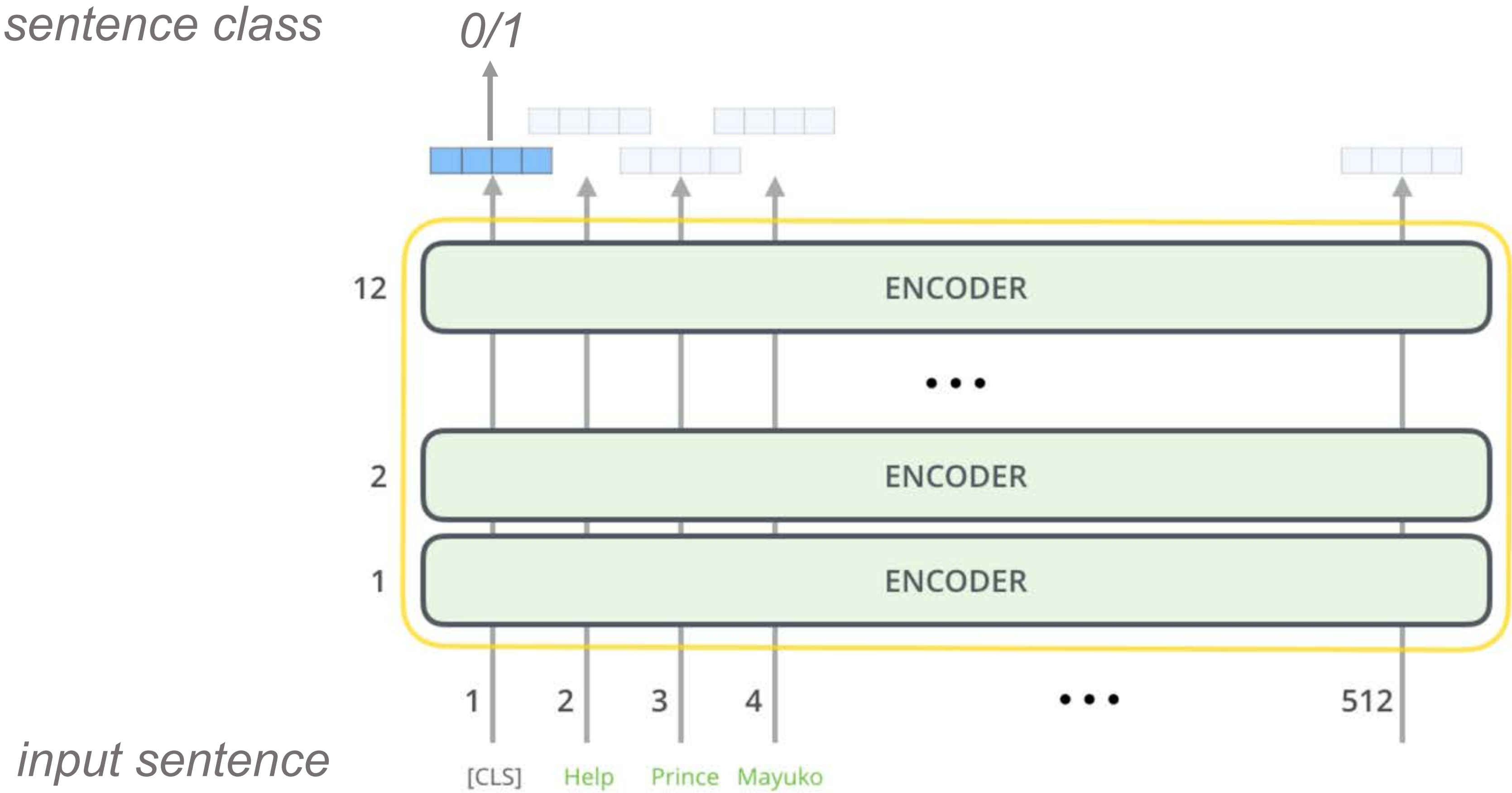


input sentence

Other Features (II): SOTA Pretrained Models

Bert [Devlin et al., 2018]

sequence classification	model = BertClassifier(hparams={'clas_strategy': 'cls_time'}) logits, preds = model(input_ids, input_length, segment_ids)



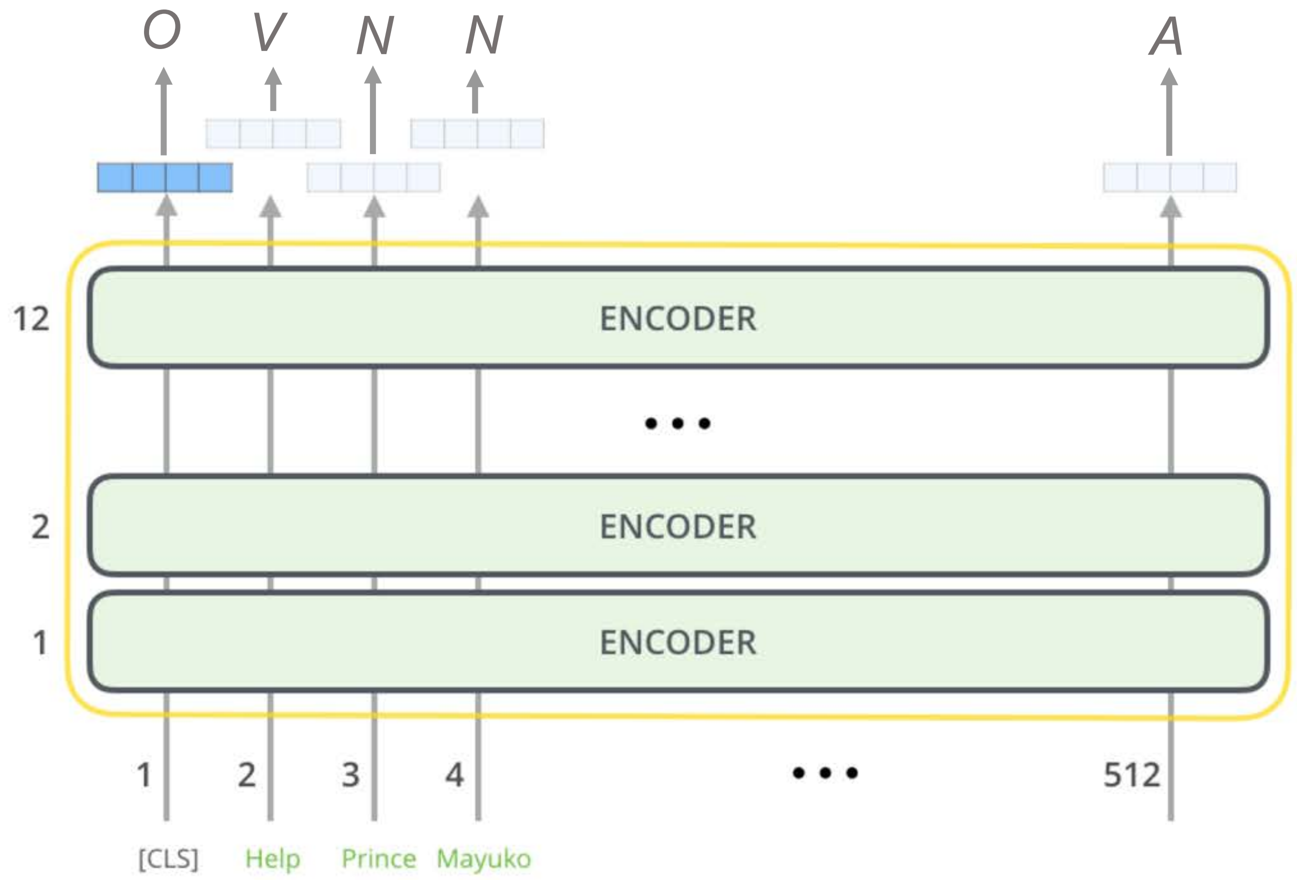
Other Features (II): SOTA Pretrained Models

Bert [Devlin et al., 2018]

sequence labeling

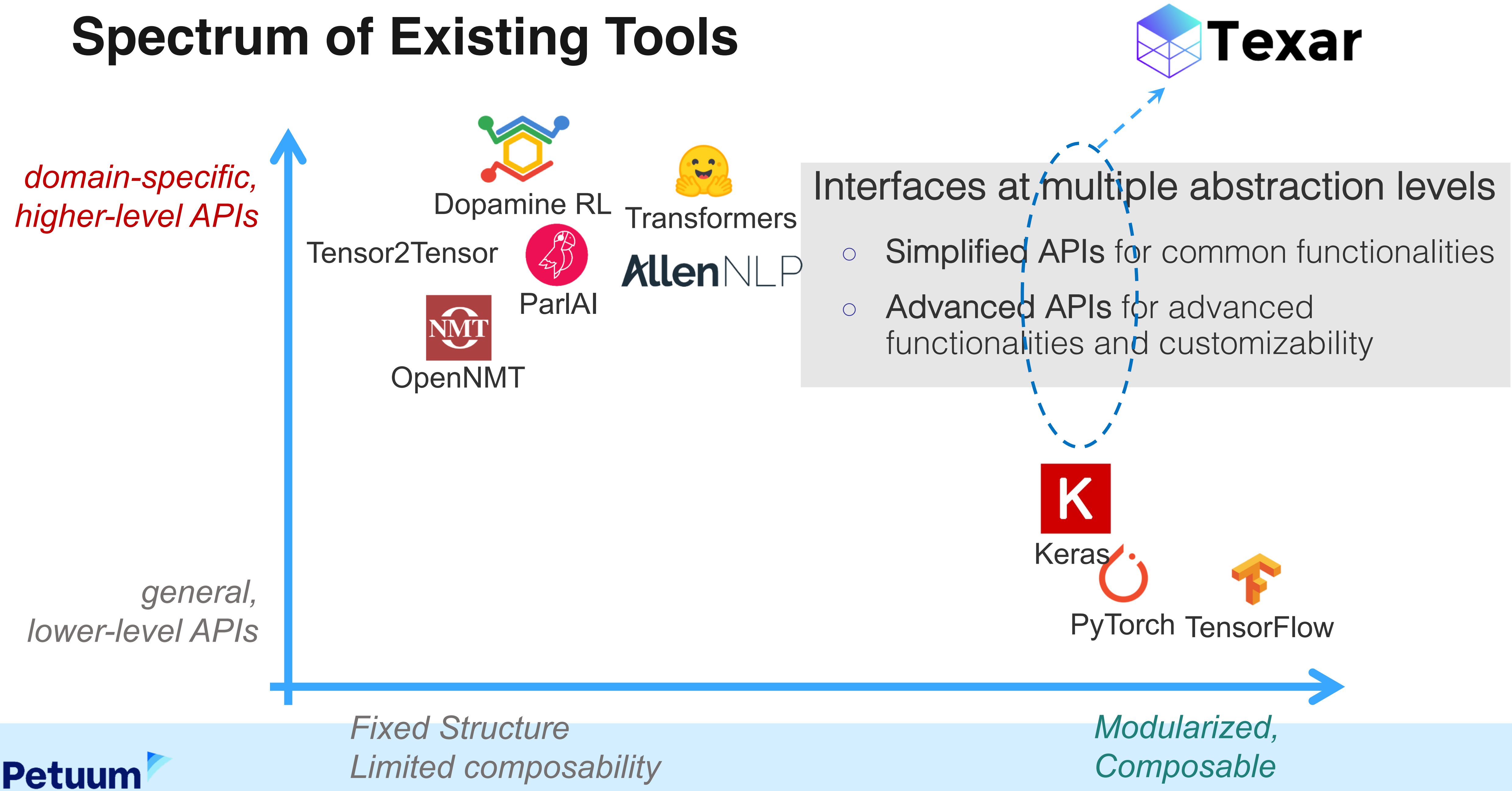
```
model = BertClassifier(hparams={'clas_strategy': 'all_time'})
logits, preds = model(input_ids, input_length, segment_ids)
```

label per step



input sentence

Spectrum of Existing Tools



Interfaces at multiple abstraction levels

- Simplified APIs for common functionalities
- Advanced APIs for advanced functionalities and customizability

Petuum

Spectrum of Existing Tools

- Texar
  - Composable ML system – create ML models of complex design from ML first principles
- TensorFlow, PyTorch
  - Symbolic languages for low-level development of ML models
- BERT, GPT2, …
  - ML models used to model human languages; also can refer to their downloadable pre-trained weights (ready-to-run)
- Amazon SageMaker, Google AutoML, Microsoft Azure ML
  - Commercial products for training, inference and management of pre-built ML models; may perform algorithm-driven model parameter or architecture tuning

Applications of Texar

Many products built on Texar

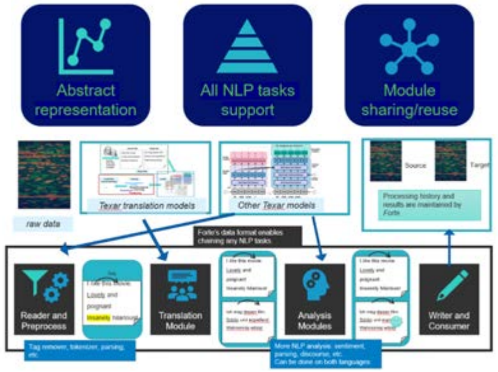
- FORTE – templates for larger complex NLP applications

- Chest X-Ray report writer
- Medical Registry report writer
- ICD coding system

- Financial knowledge base builder
- Financial summary/report writer

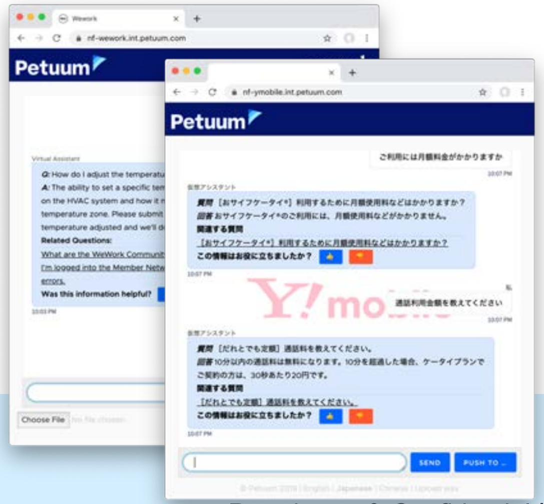
- Multi-Lingual Cognitive Chat Bots
  - For Call Center Support
  - For Retail In-Store Assistance

FORTE

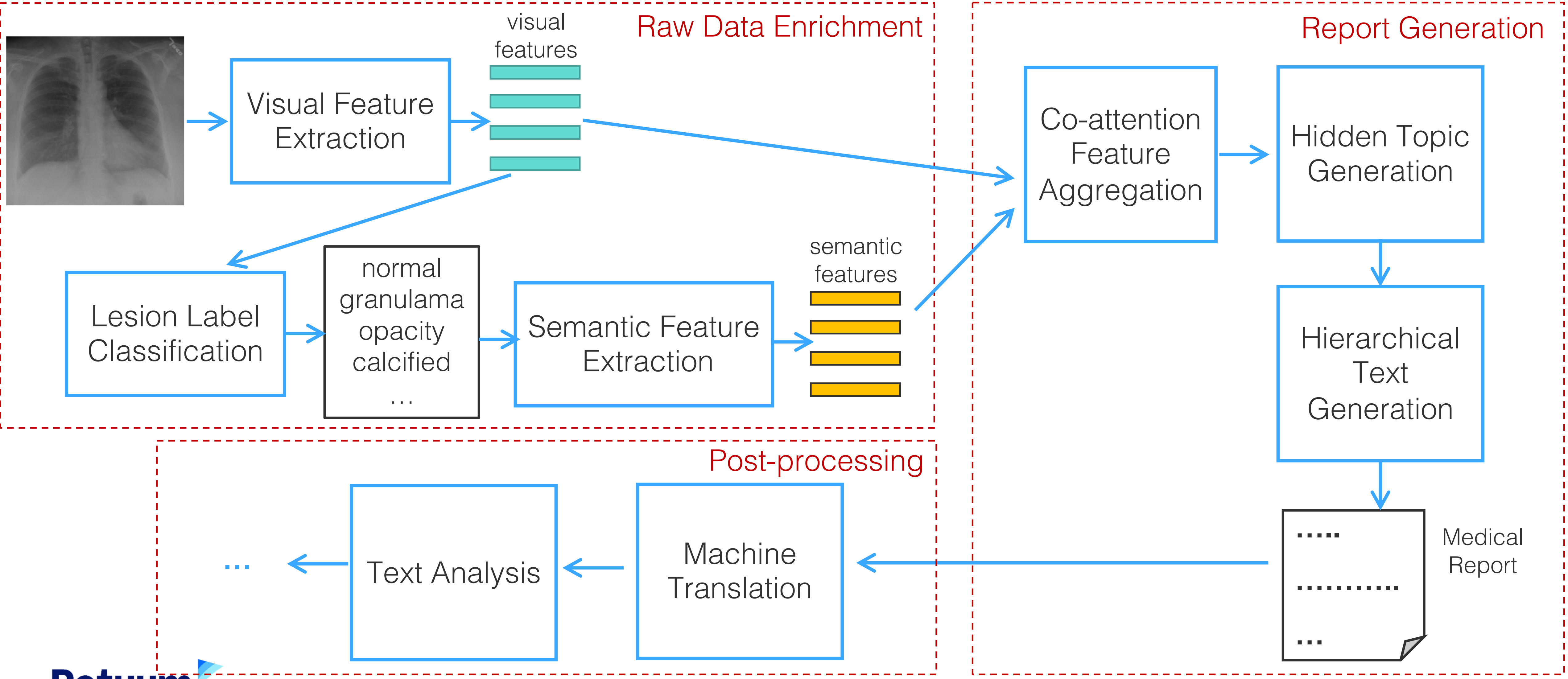


Chest X-Ray Report Writer

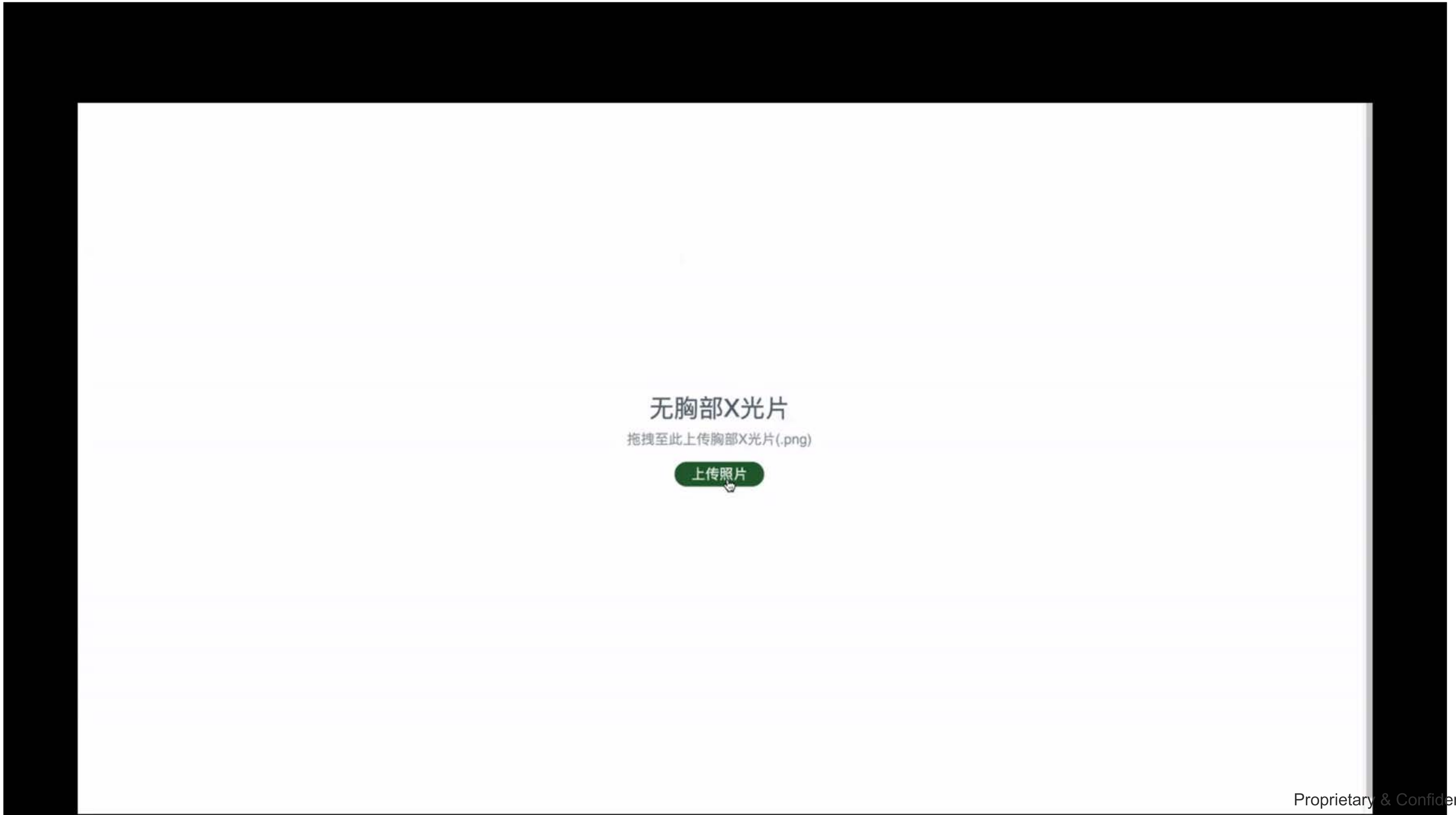
Multi-Lingual Cognitive Chat Bots



Example: Chest X-Ray Report Writer



Example: Chest X-Ray Report Writer + Translation



Texar Resources

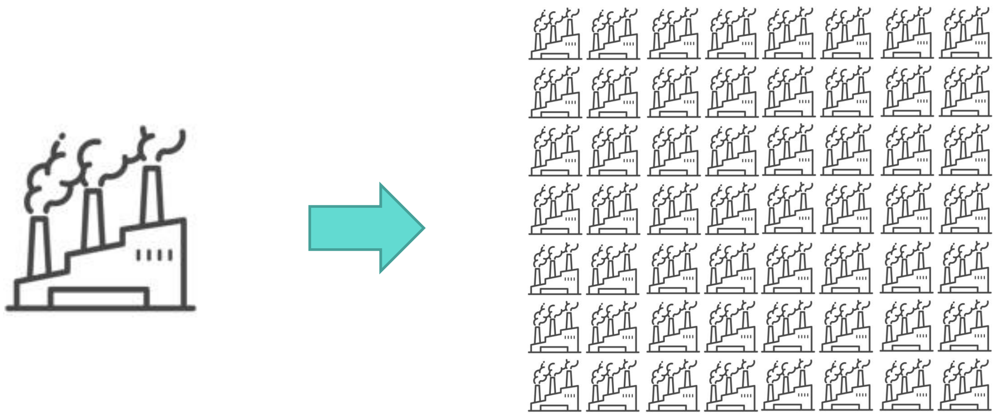
Texar-TF

Texar-PyTorch

- Website: https://asyml.io
- GitHub (TF version): https://github.com/asyml/texar
- GitHub (PyTorch version): https://github.com/asyml/texar-pytorch
- Examples: https://github.com/asyml/texar/blob/master/examples
- Documentation: https://texar.readthedocs.io/
- Blog: https://medium.com/@texar
- Tech report: https://arxiv.org/pdf/1809.00794.pdf

Scalable AI Infrastructure

Scaling out AI applications – easier, faster, cheaper



What is needed to scale AI?

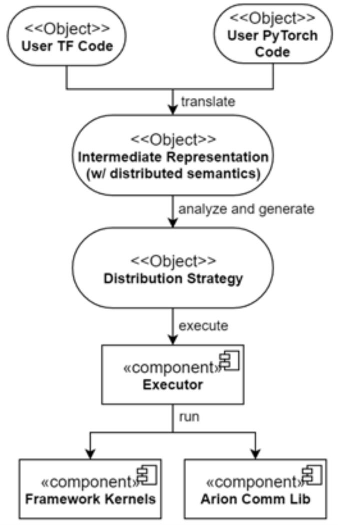
Inter-Task Interfacing Application Templates
Assemble many tasks with uniform interfaces to form larger, complex applications

High Speed & Scalability
Made-to-order, just-in-time distributed training strategies

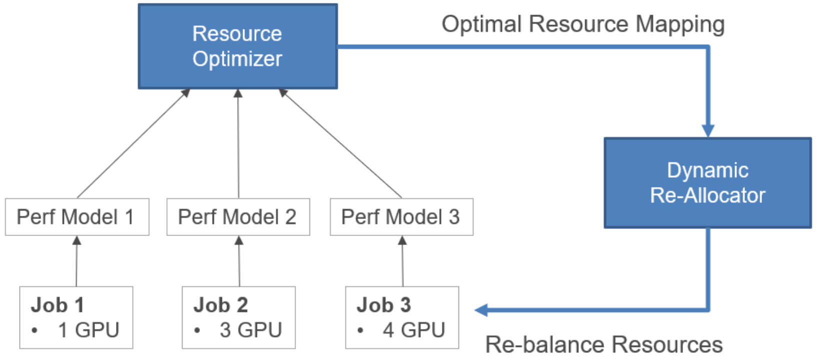
Save Cost & Time
Dynamically reassign CPU & GPU resources for fastest/cheapest workload completion time

FORTE

ARION

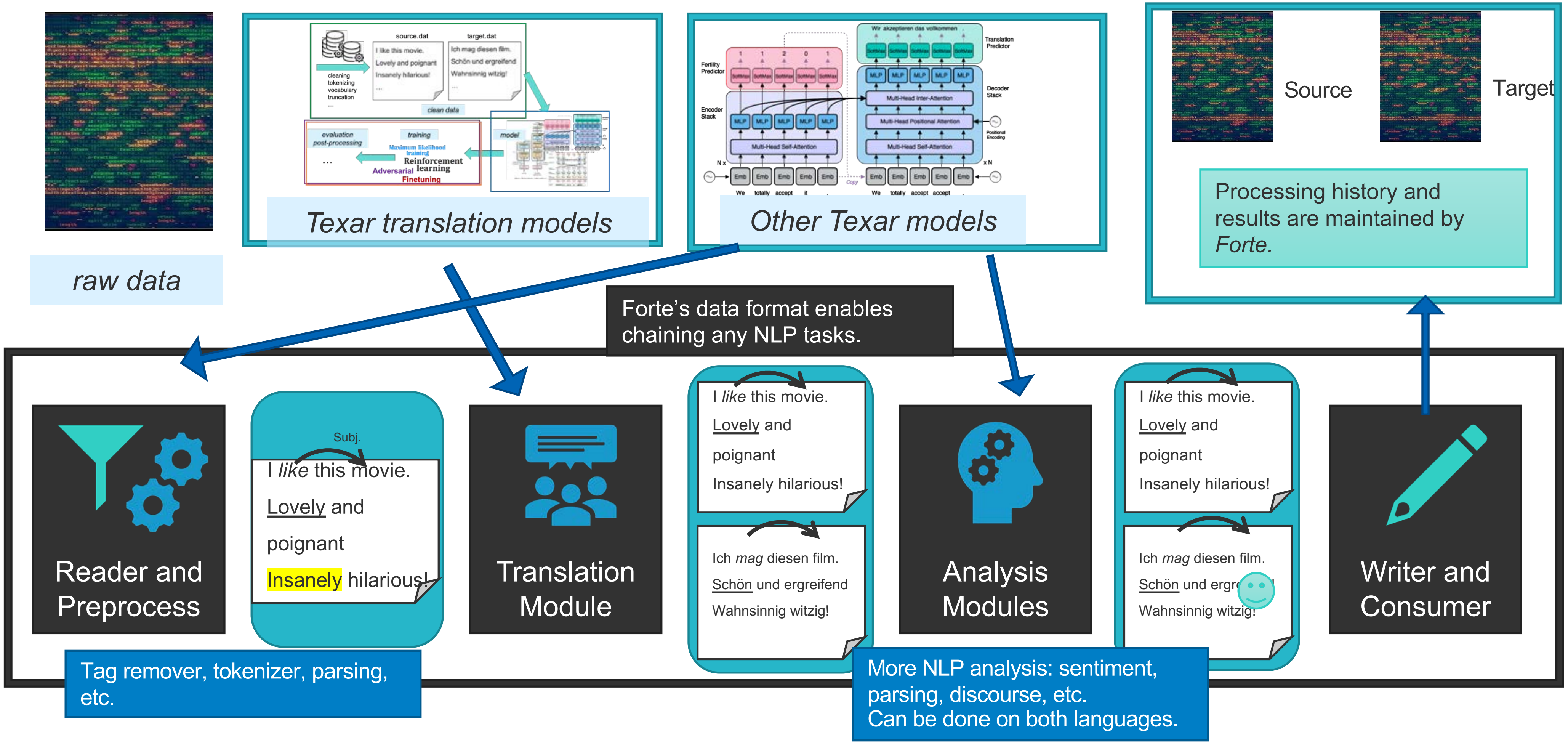


ESPER



Forte – Flexible Scaffold for Larger NLP Applications

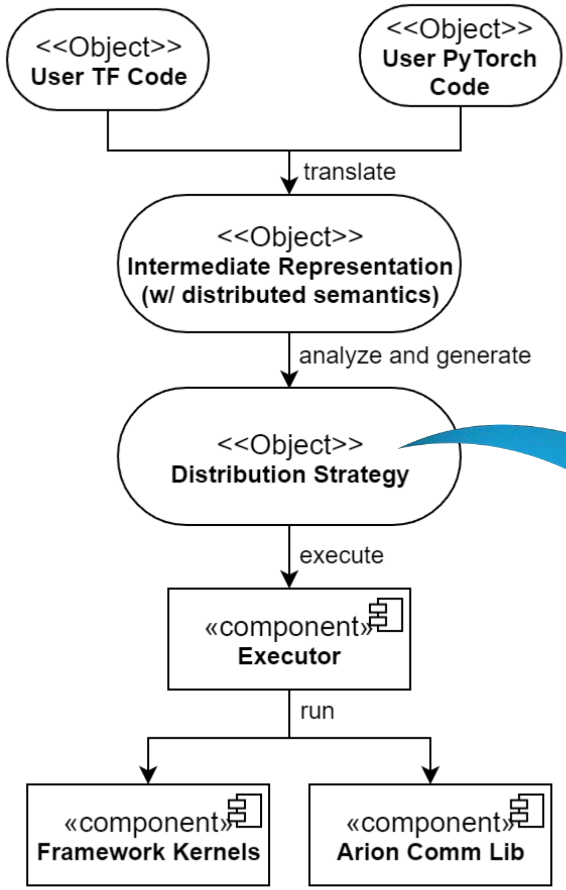
Built upon Texar



Petuum

Arion – High Speed, Scalability for ML Model Training

Tailor-made distributed training strategies for hard-to-scale ML models



Pain Points

- Existing systems are mostly specialized and implemented based on 1 (probably at most 2) of those system architectures.

AR | PS | SSP | SFB | Decent-ralized | Gradient Compression

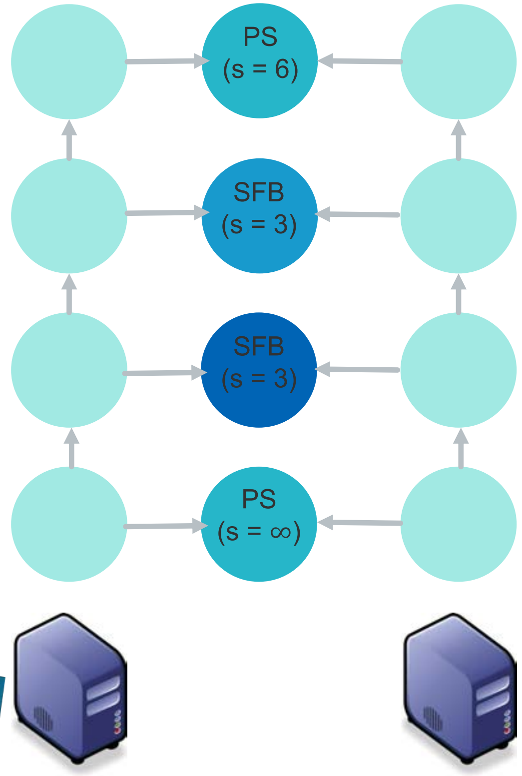
Pipelining (NN) | Horovod (AR) | Poseidon (PS+SFB) | Dynamic Batching

- Distribution performance/scalability is only achieved under certain condition, for a narrow family of models where the system is specialized.

Solution

Arion expresses models and training algorithms as an intermediate representation (IR) with distributed execution semantics

Arion rewrites IR and generates **tailor-made distributed execution strategy** using multiple system architectures (e.g. PS + SFB)



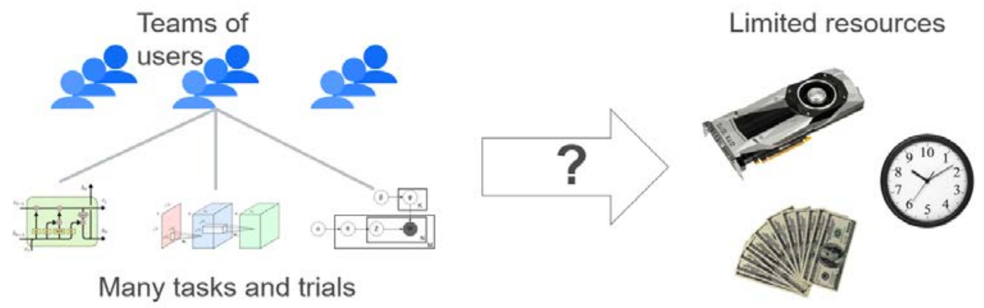
Value: Model training – from un-scalable (speed ∝ 1) to highly-scalable (speed ∝ *n* machines)

Arion is 1.1-1.5x faster vs Parallax (SOTA) | 2-5x faster vs TF | 4-9x faster vs Horovod

Esper – Save Compute Costs and Time

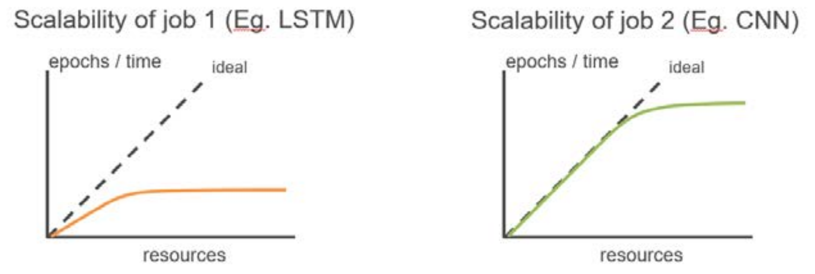
Dynamically reassign CPU/GPU to the most impactful work

Pain Points



How to best share limited resources between many ML jobs?
Can a software infrastructure automate resource allocation for ML?

Does fairness-based resource allocation work well for ML? Consider:

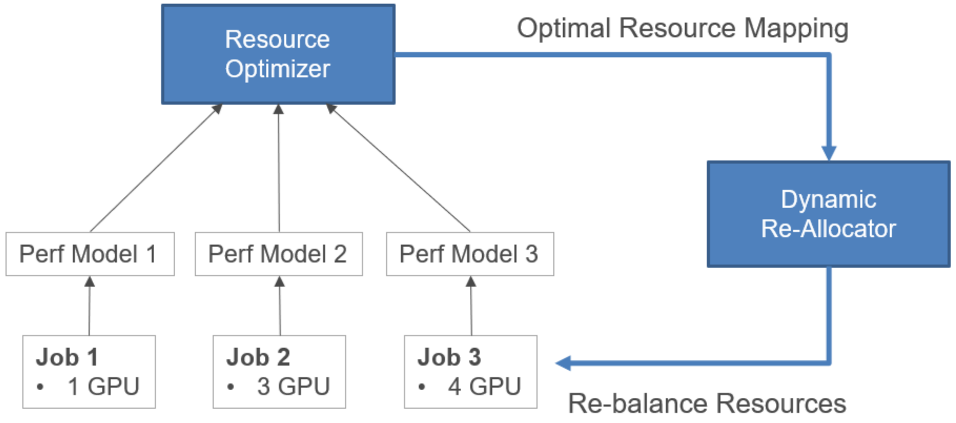


Is it best to give the same amount of resources to these two jobs? NO.

Solution

Esper learns the performance model for each model training job

Esper uses performance models to optimize (in near-real-time) CPUs/GPUs needed by each running job on the cluster/cloud



Value: Finish training *multiple* models faster (and cheaper!)

Esper is 2-4x faster vs Google Kubeflow

Petuum

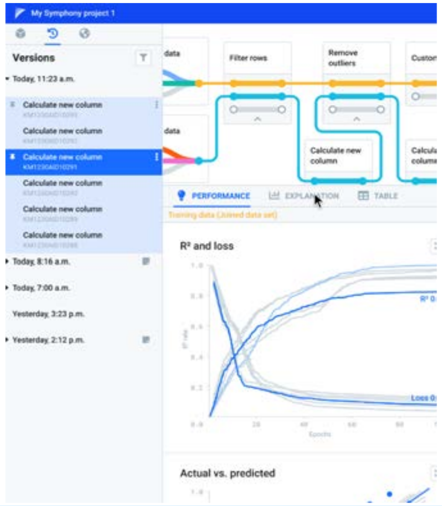
What is needed to productize AI?

Drag & Play UI
Manage, Experiment, Version
Systems, Tools, Infra,
Visualizations, Dashboards

Programming Language InterOp
Integrate & Manage different code
(C++, Python, Scala, TF, PyT, Spark, …)
all into the same ML app

Infra Management
Authentication/Security
Containers
High-Availability
Distributed & Cloud Storage
Cloud & On-Premise Deployments

COMPOSER



FUGUE

doubler.py

```python
import fugue.processor

@fugue.processor.register(declaration="doubler.yml")
def double(data, models, conf, column_selectors):
    input_gen = data['Input Dataset']['Data']
    selected_column = column_selectors['Input Dataset']['to_double'][0]

    def gen():
        while True:
            dataset, iter_info = next(input_gen)
            output_column_name = f'{selected_column}_doubled'
            dataset[output_column_name] = [float(x * 2) for x in dataset[selected_column]]
            yield dataset

    return {'Datasets': {'Output Dataset': gen()}}
```

CHIRON

AI (Civil) Engineering with Petuum

Industry Agnostic

Robo-radiologist

Insurance Auto-Report

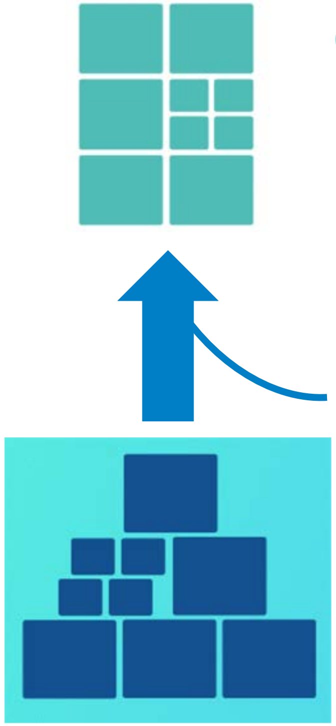
Virtual EA

Smart Expense Reports

Smart Catalog

Robot Store Staff

Building AI Like Lego

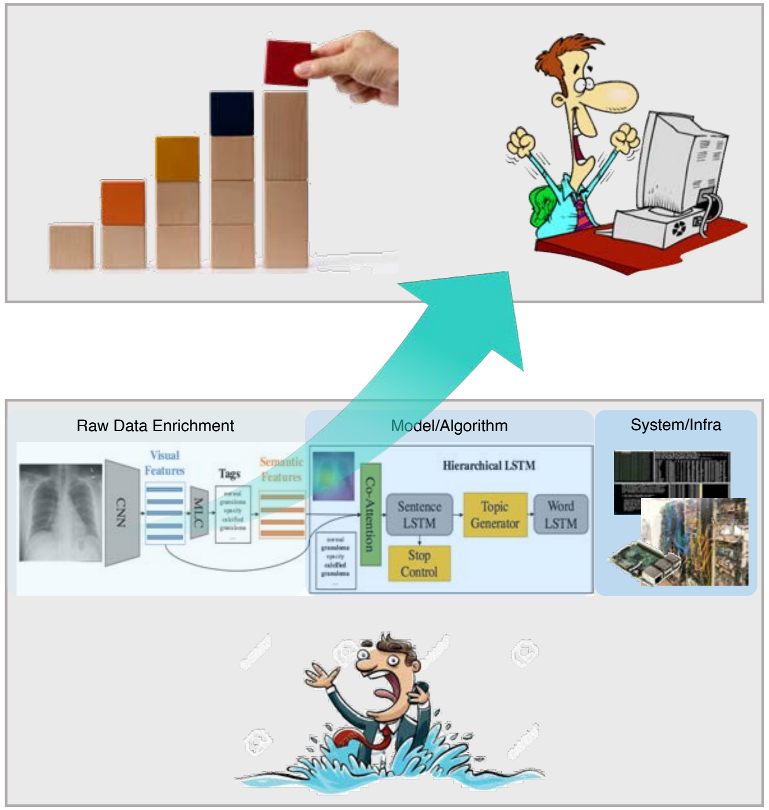


Completed software

10% White-Glove Assembly

90% Completed building blocks

AI With No Tears



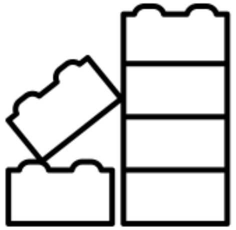
Petuum OS

PETUUM

Petuum's Mission

Industrialize AI technology
– turning it from black-box artisanship into standardized engineering process

Transform enterprises across industries
– turning them into owners, builders, and informed users of AI



Sustainable & Standardized Building Blocks

One foundation for your current and future AI-building needs

A 2018 WEF Technology Pioneer Winner

WE ARE HARING:

ML engineer/manager
Software engineer/manager
System engineer
UI engineer
…

PETUUM

Thank You

Carnegie Mellon University School of Computer Science